

MULTI Integrated Development Environment 5.3x

(Supports the Cafe SDK)

Version 1.06

Advanced Data Controls Corp.



Copyright 2011 Green Hills Software, Inc. & Advanced Data Controls Corporation. All rights reserved.

**The content of this document is highly confidential
and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

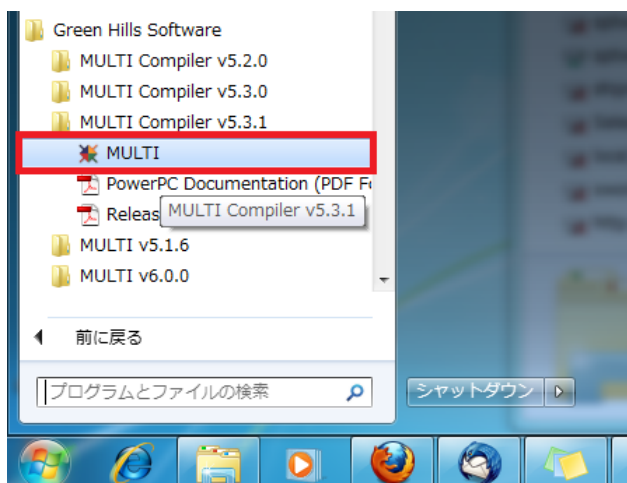
Table of Contents

1	Creating a New Project (for Cafe SDK)	4
2	Modifying a Project (for Cafe SDK).....	9
2.1	Modifying Linked Libraries	9
2.2	Creating Debug/Release Projects	14
3	Debugging Programs	22
3.1	Launching the MULTI Debugger	22
3.2	Launching the MULTI Debugger from a Cygwin Shell.....	26
3.3	Breakpoints	27
3.4	Displaying Registers	28
3.5	Displaying Memory	31
3.6	Displaying Data.....	32
3.7	Displaying the Call Stack	33
3.8	Returning from Callees	34
3.9	Switching Between Source and Assembly Views	35
3.10	Browsing Cross References	36
3.11	Browsing Component Source Files and Functions.....	37
3.12	Reloading Programs	39
3.13	Restarting Programs	40
	Revision History	41

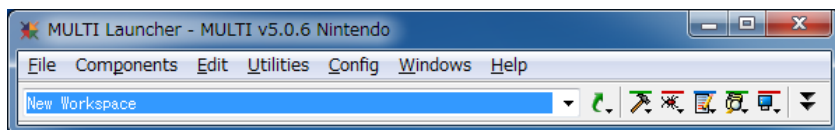
1 Creating a New Project (for Cafe SDK)

This chapter explains the basics of how to create new projects for the Cafe SDK using the MULTI integrated development environment made by Green Hills Software.

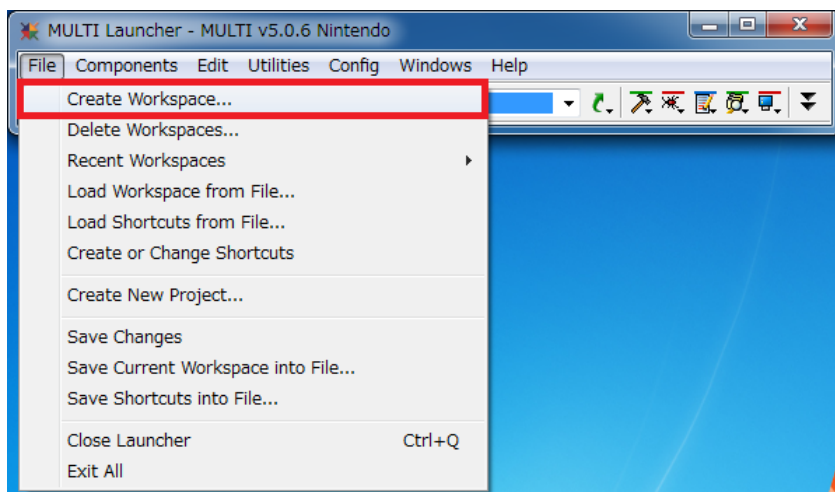
1. To open MULTI, in the Windows Start Menu, select **All Programs > Green Hills Software > MULTI Compiler v5.3.x > MULTI**. (In this example, version 5.3.1 is launched.)



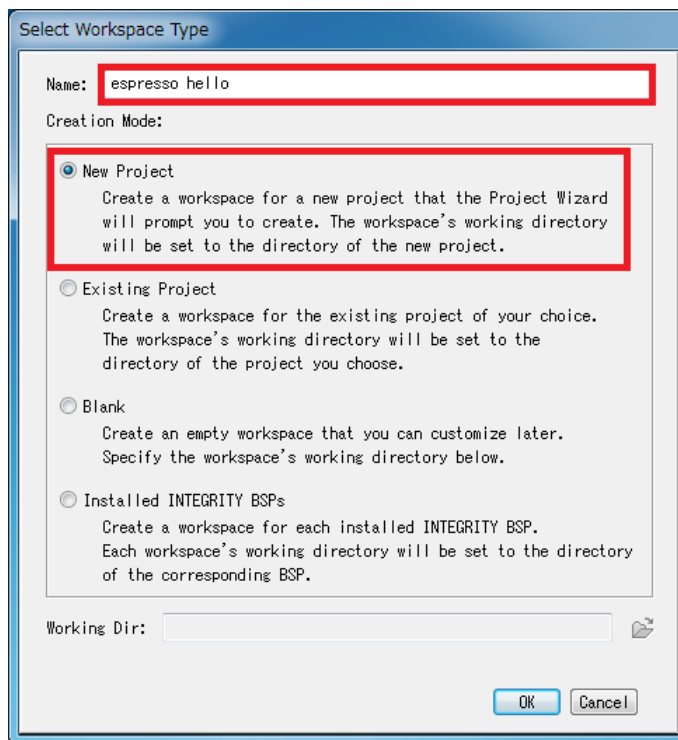
2. This opens the MULTI Launcher.



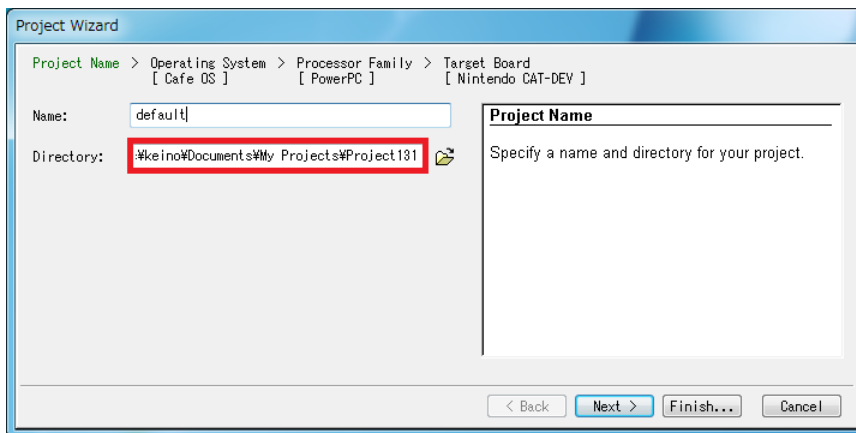
3. Select **File > Create Workspace**. Alternatively, you can click the **Run Action Sequences or Shortcuts** button and select **Create Workspace**.



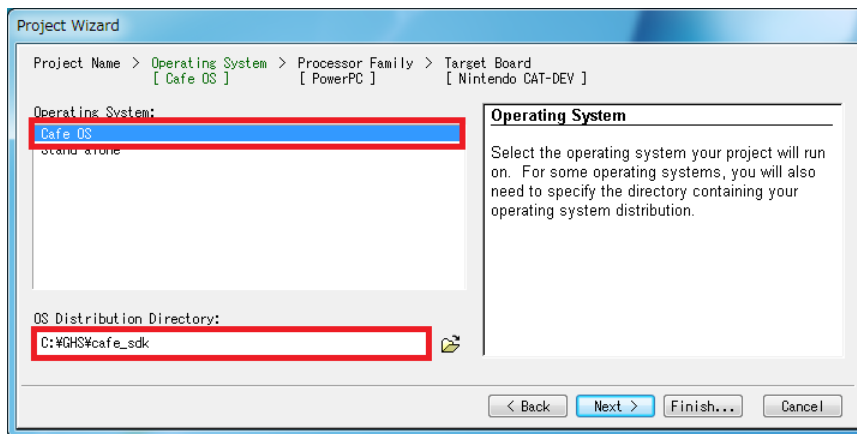
4. The **Select Workspace Type** dialog box will appear. Enter a name for your workspace, and choose the **New Project** option. Click **OK** to continue.



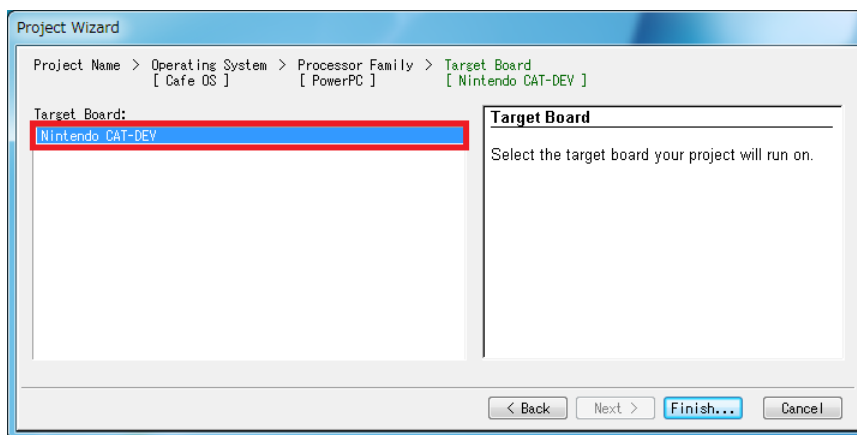
5. The **Project Wizard** will appear. With the default settings, a new folder will be created in the **My Projects** sub-directory within the documents folder for the active user. The Project Wizard will assign the names "Project1," "Project2," and so on by default. Enter a name for your project, and specify a file path. Click **Next** to continue.



6. Choose the operating system. To create a project using the Cafe SDK, choose **Cafe OS**. (If you won't be using the SDK, choose **Stand-alone**.) Click **Next** to continue.
- Enter the path to your Cafe SDK installation (for example, c:\GHS\cafe_sdk) in the **OS Distribution Directory** field. Click **Next** to continue.

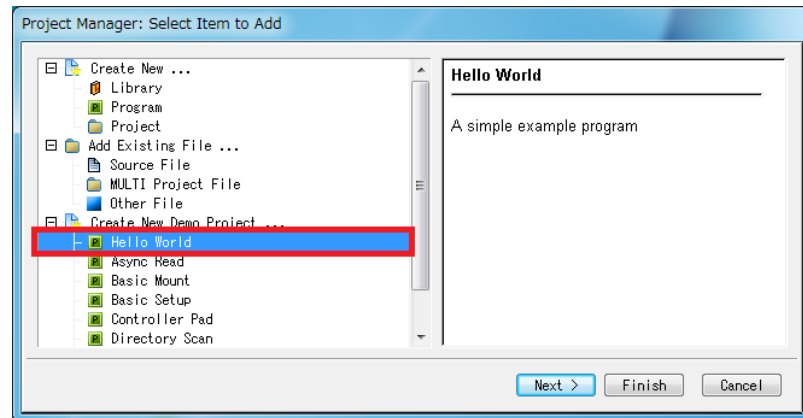


7. Under **Target Board**, select **Nintendo CAT-DEV**. Click **Finish** to continue. Depending on your operating system and security settings, a Windows Security Alert may appear asking if you want to keep blocking the MULTI Software Development Environment. Click **Unblock** if prompted.

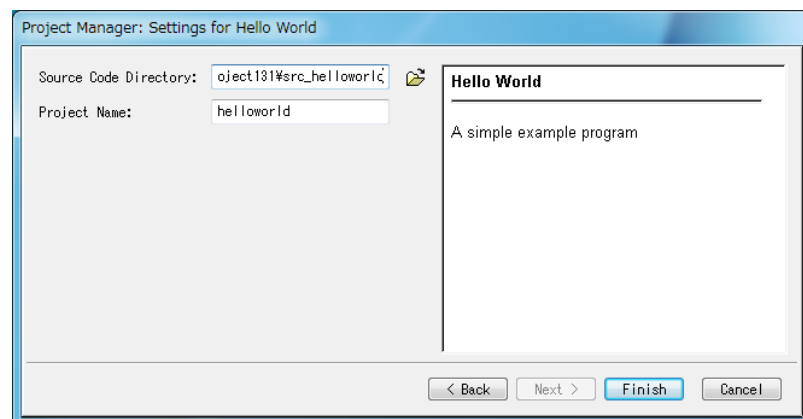


8. Next, you'll need to choose a type of project.

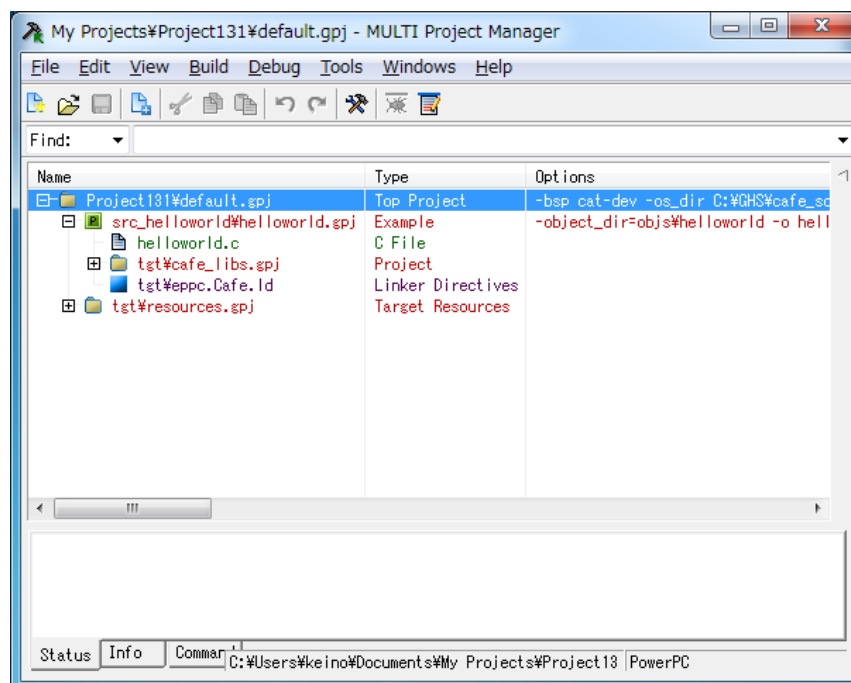
In this demo, we'll try creating a "Hello World" application. Under **Create New Demo Project**, select **Hello World**, and then click **Next**. Later, when you want to create your own programs or libraries, select **Program** or **Library** (respectively) under **Create New**.



9. Enter a project name. The default values will work just fine. Then click **Finish** to continue.



10. The new project will be created automatically.

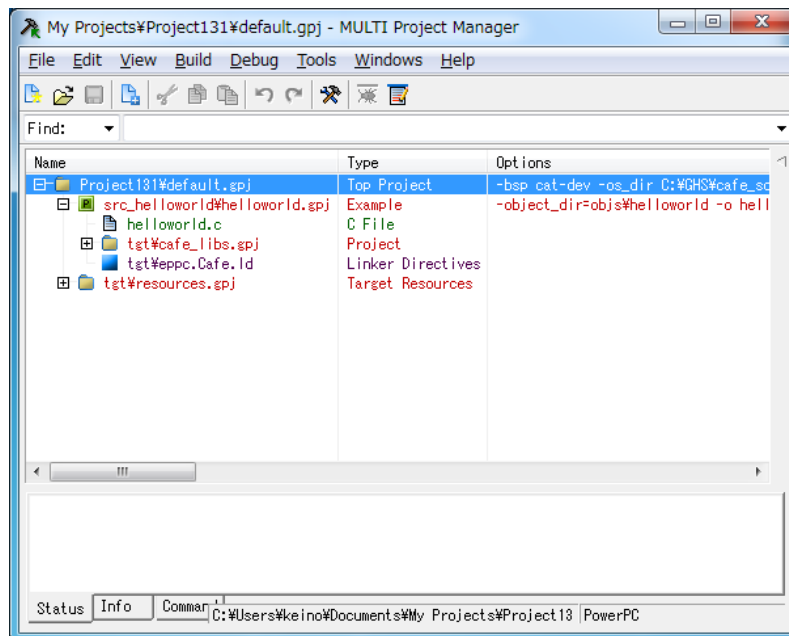


2 Modifying a Project (for Cafe SDK)

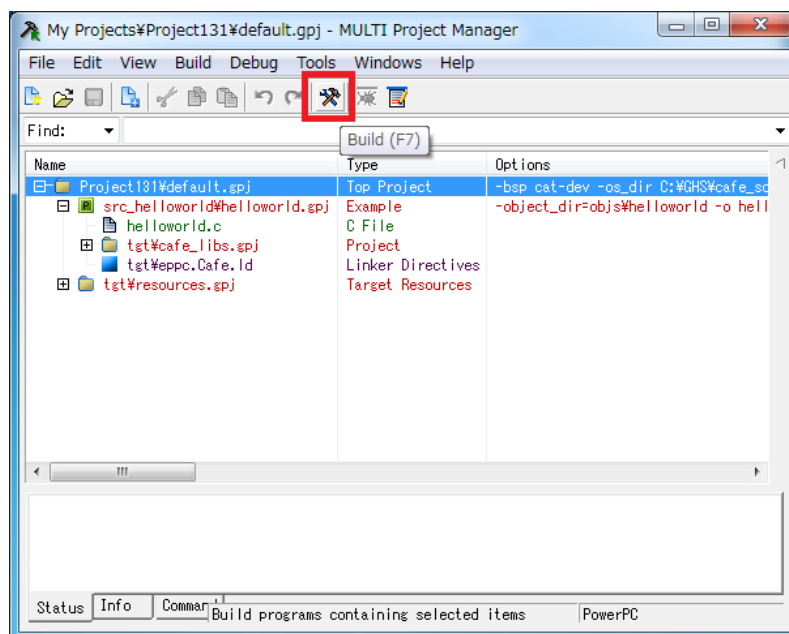
This chapter explains how to modify existing Cafe SDK projects that were created using the MULTI integrated development environment made by Green Hills Software. (The examples in this document were made using SDK 1.4.1.)

2.1 Modifying Linked Libraries

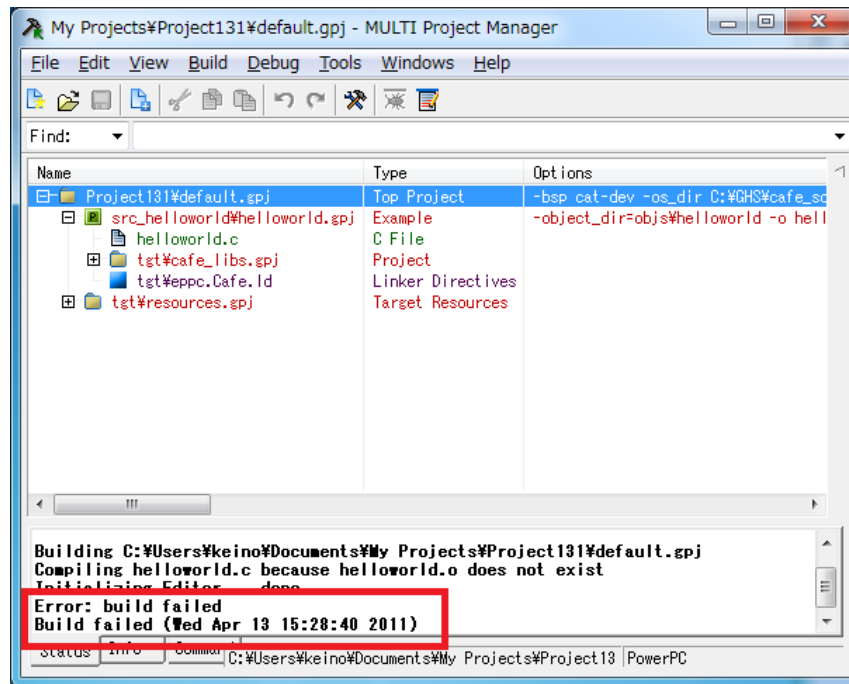
1. Open a Cafe SDK project.



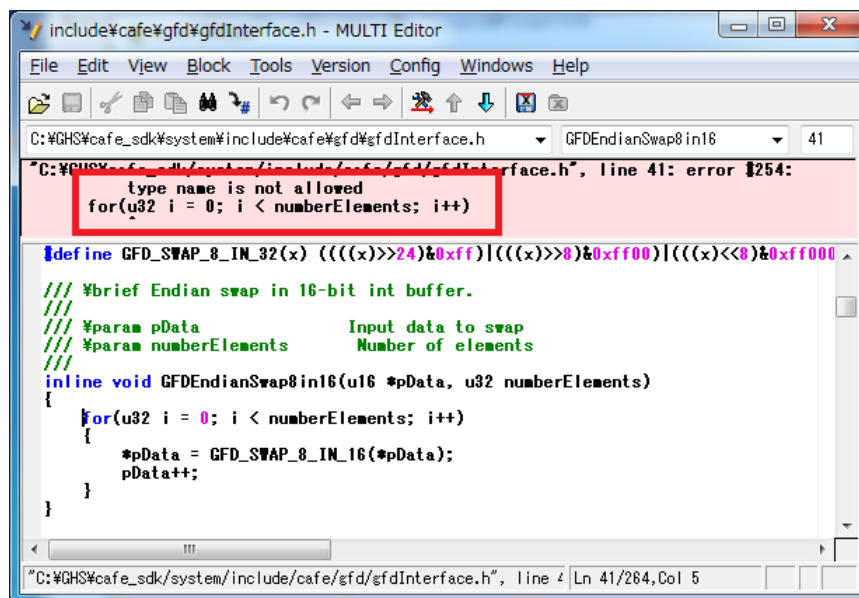
2. First, try to build the project. This can be done by clicking the **Build** button or by pressing F7.



3. If the build fails, the error message "build failed" will be displayed, and the source file that caused the compilation error will be opened automatically.

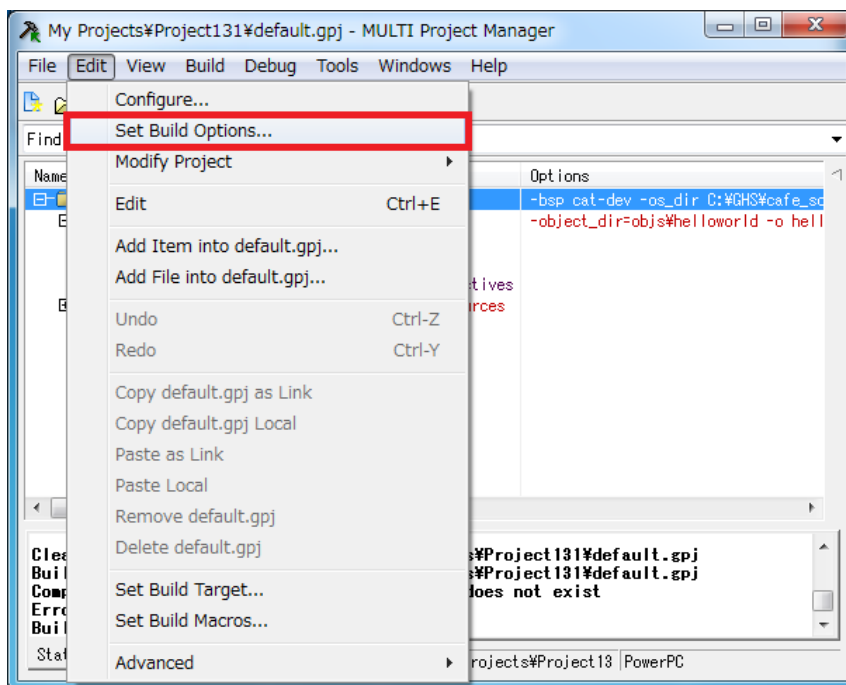


For example, in the example error below, the error message "type name is not allowed" is displayed because the declaration of variables within a *for* statement is not allowed in ANSI C.

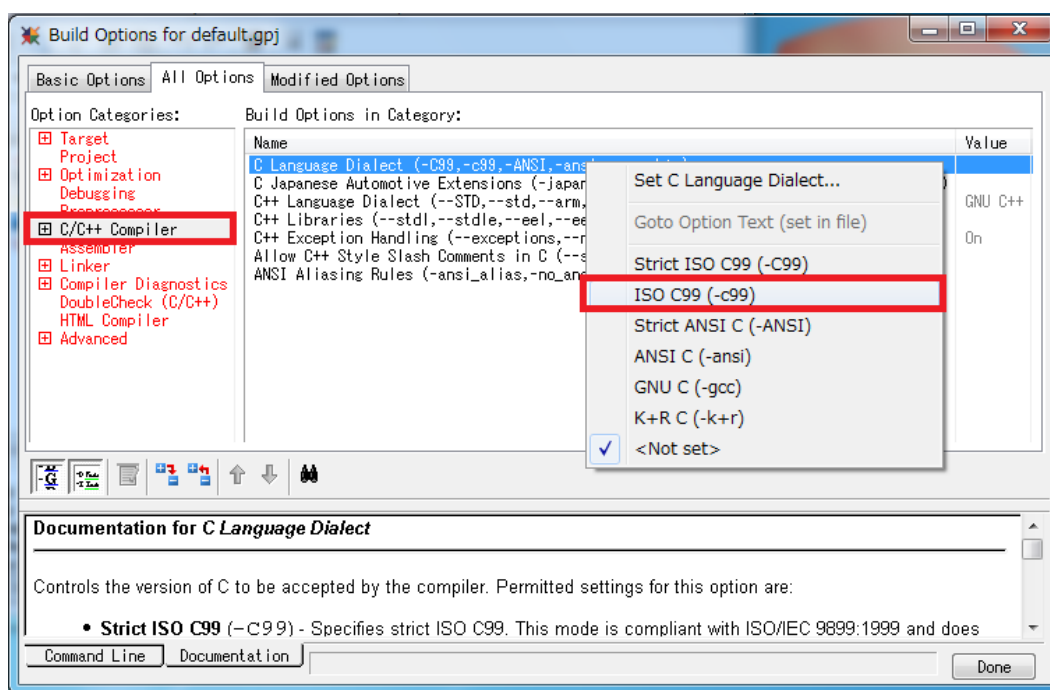


4. To fix this, we'll change the C-language specification to "ISO C99" under the compilation options. The build options for the Cafe SDK are all grouped into the file `cos.opt`. To make the necessary change, either add the `-c99` option to this file, or change the project's options using the procedure shown below.

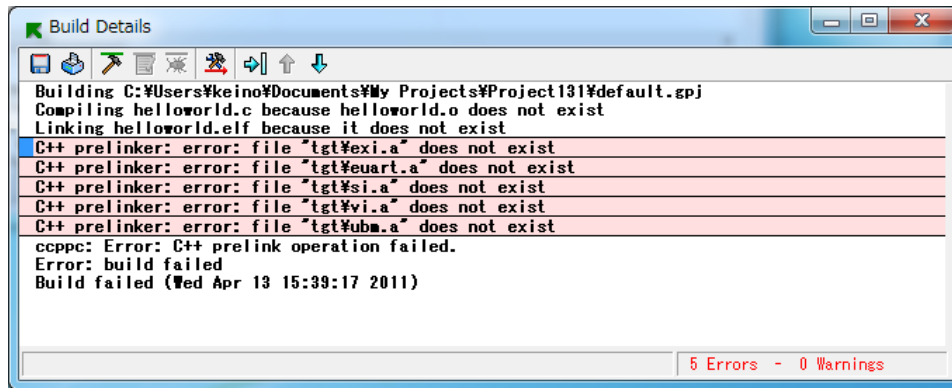
5. Select the top project (default.gpj in the example below), and then select **Edit > Set Build Options**. Alternatively, you can double-click the **Options** column if you've displayed it (it's hidden by default).



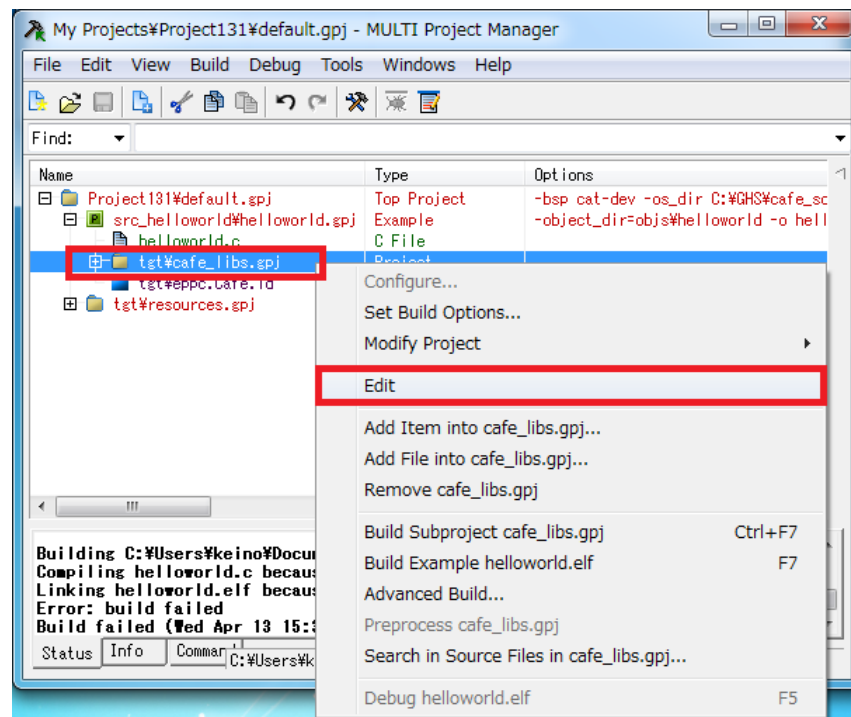
6. This will display the **Build Options** window. On the **All Options** tab, select **C/C++ Compiler**, set the **C Language Dialect** option to **ISO C99**, and then click **OK**. Click **Done** to continue.



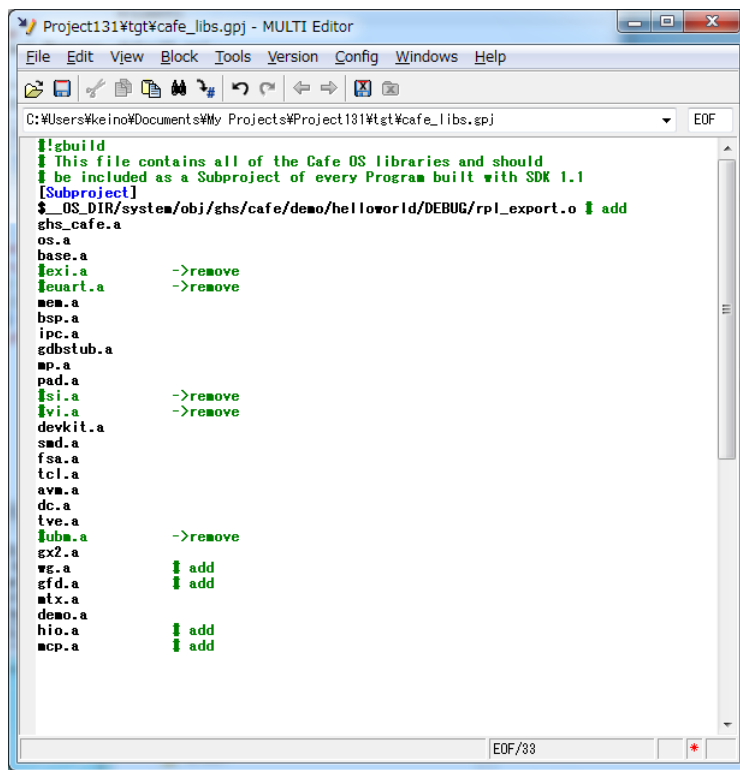
7. Rebuild with the new options.
8. If the version of the Cafe SDK and MULTI you are using link against differing libraries, you may see errors stating that certain libraries could not be found. An example is shown in the following image.



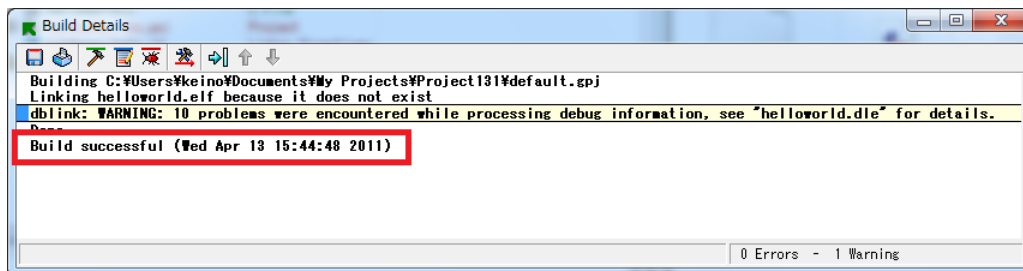
9. If this type of error occurs, right-click `tgt\cafe_libs.gpj`, click **Edit**, and make the necessary edits. If you are simply removing unwanted libraries, you can also select the libraries you wish to remove in your project window, right click, and select "remove selected items." This will open the MULTI editor.



10. Edit the file so that your project will link against the appropriate libraries for the version of Cafe SDK that you are using.



11. The *default* list of libraries to link against is stored in the project file shown below (the path below indicates the relative path from your MULTI installation folder). If you edit this file to contain all libraries your projects need to link against, it will be copied into any new projects you create thereafter. defaults\new_project\components\os_cos\cafe_libs.gpj
12. After making this edit, rebuild your project once again by pressing F7. If the message *Build successful* is displayed, the build has completed. In the screenshot below, *dblink*-related warnings have occurred; these warnings can be ignored.

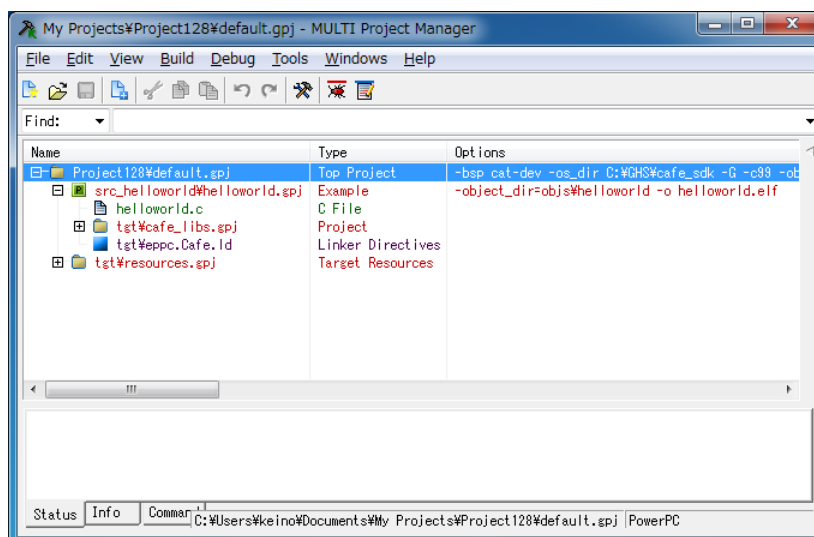


13. The executable file helloworld is now complete.

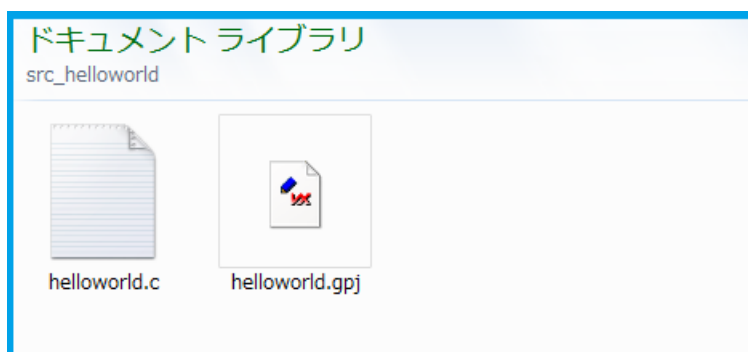
Note: The MULTI package includes an Eclipse plug-in that allows you to use the Eclipse environment to manage your projects, configure your options, build (make) your code, emulate instruction sets, and more. In this scenario, the MULTI debugger is launched when you select an option from Eclipse's **Debug** menu.

2.2 Creating Debug/Release Projects

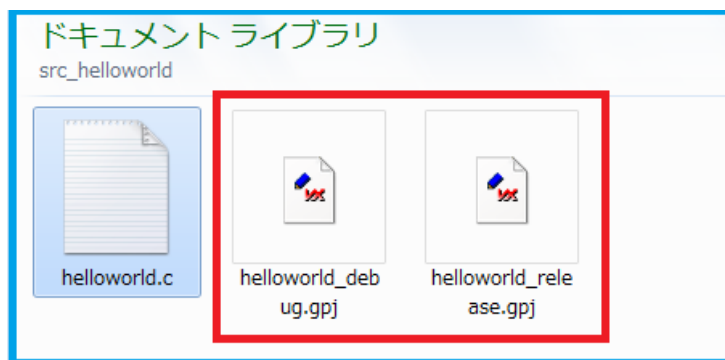
1. Create a new project in MULTI.



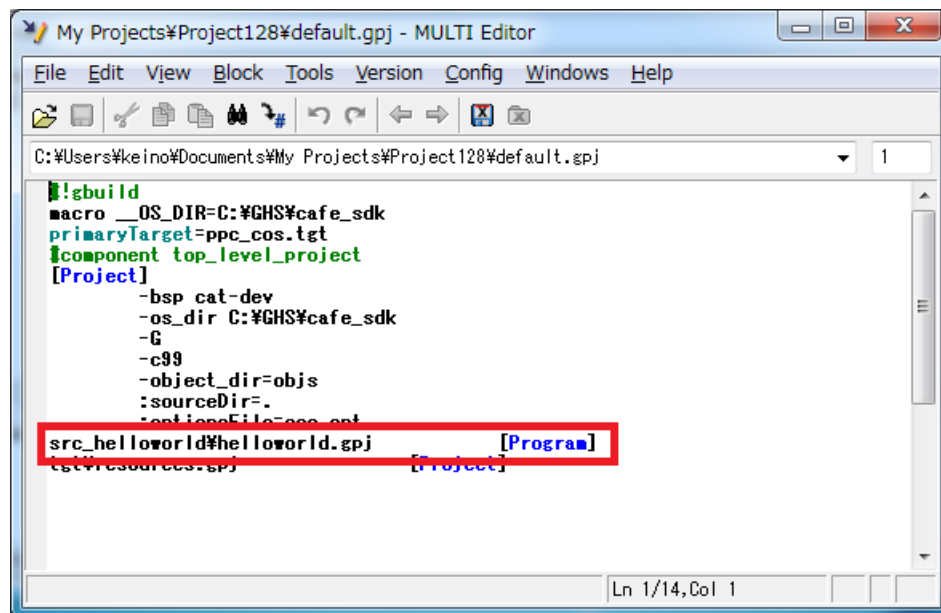
2. Open the src directory for your project (src_helloworld in the example below) in Windows Explorer.



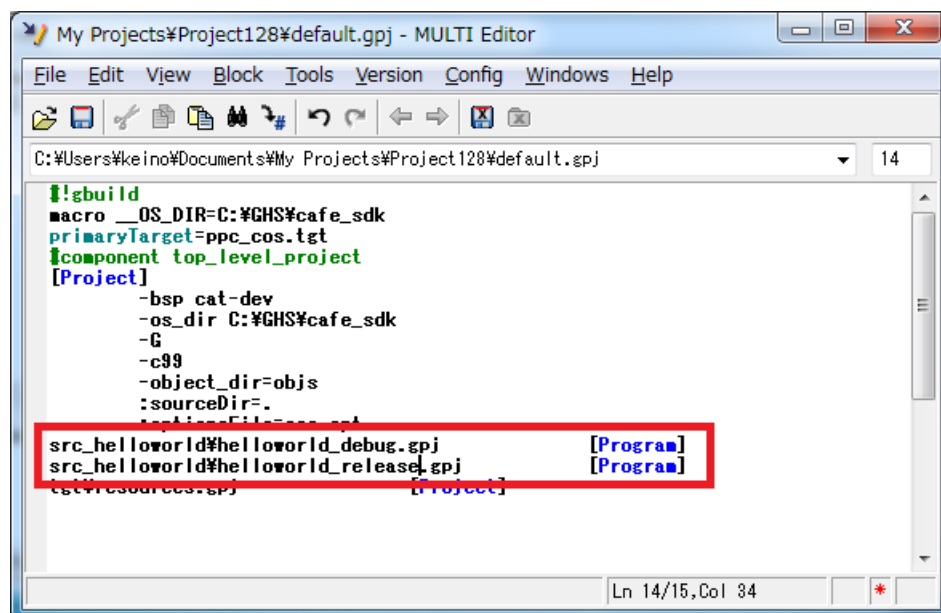
3. Make a copy of a subproject file (it should be of the **[Program]** type). Rename one for *debug* and rename the other for *release*.



4. Edit the top project file (default.gpj in this example).



5. Within the definition of the top project file, rename the entries for the two projects you just renamed.

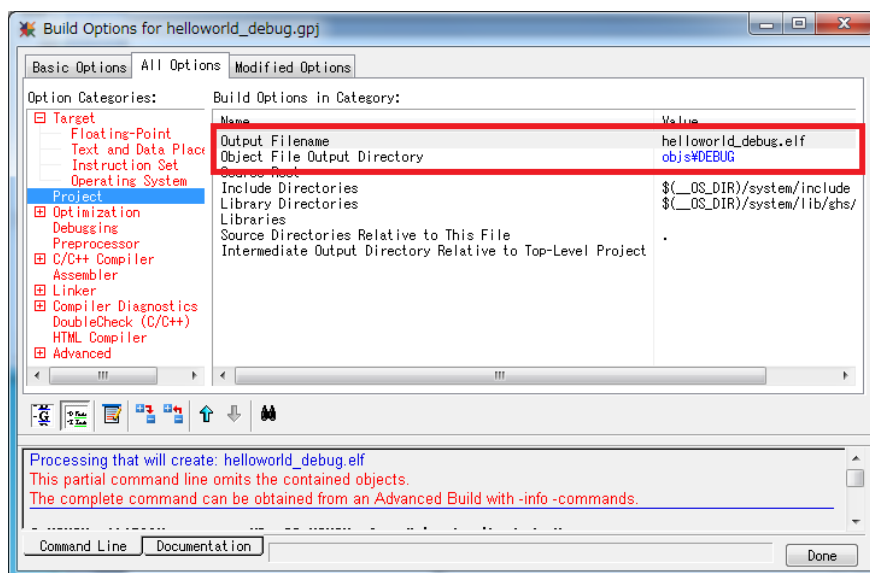


6. In the **Project Manager** window, select **File > Reload Saved Project** to apply the results of the edits you made.
7. Configure the build options for the debug project (*helloworld_debug.gpj* in this example). Right-click this file within the **Project Manager** window, select **Set Build Options**, and configure the necessary options.

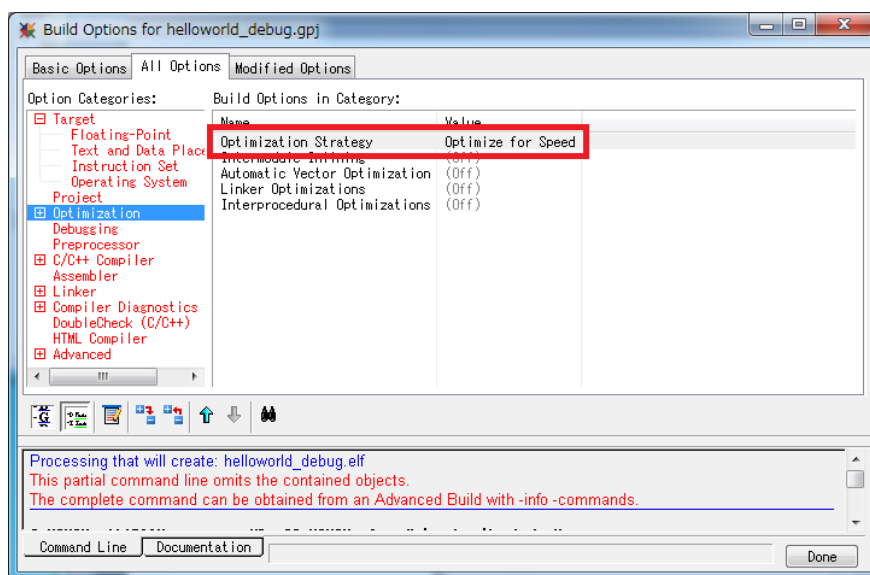
8. Some sample settings are shown below. In this example, we will add the following options.

<code>-object_dir=objs\DEBUG</code>	Directory where the DEBUG objects should be created
<code>-o helloworld_debug.elf</code>	Change the name of the ELF file that is generated
<code>-Os</code>	Optimize for speed
<code>-G</code>	Generate MULTI debugging information
<code>-D__DEBUG__</code>	Define the <code>__DEBUG__</code> preprocessor symbol

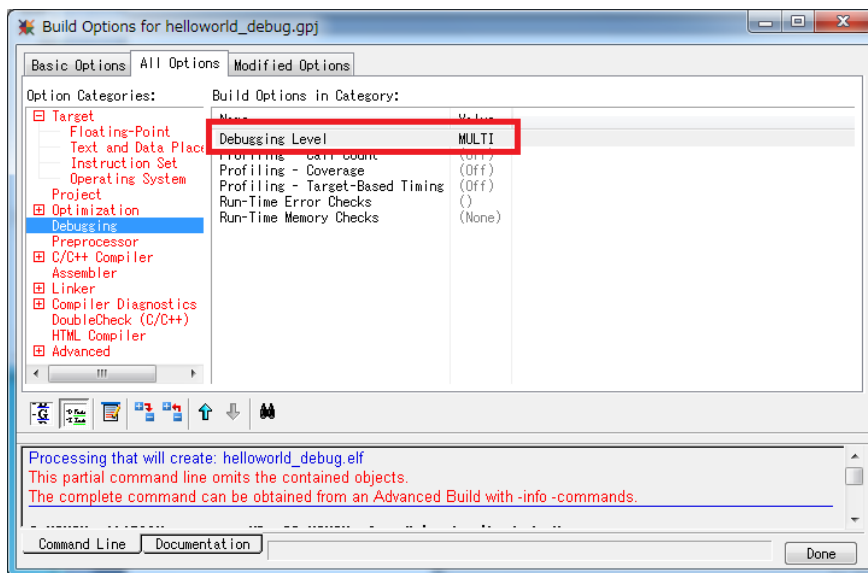
9. In the **Project** category, set the **Output Filename** option to `helloworld_debug.elf`. To make the objects be created in their own directory, change the **Object File Output Directory** option to `objs\DEBUG`.



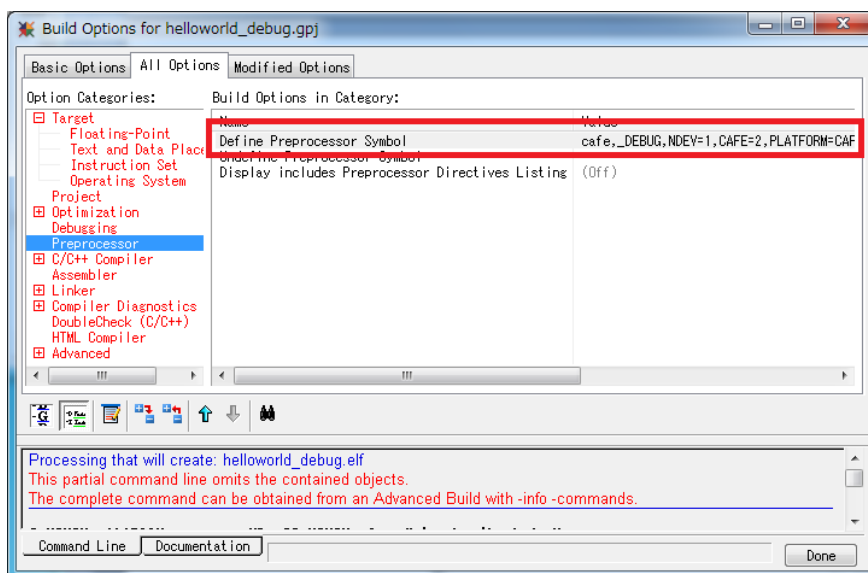
10. In the **Optimization** category, set the **Optimization Strategy** option to **Optimize for Speed**. This will prioritize execution speed.



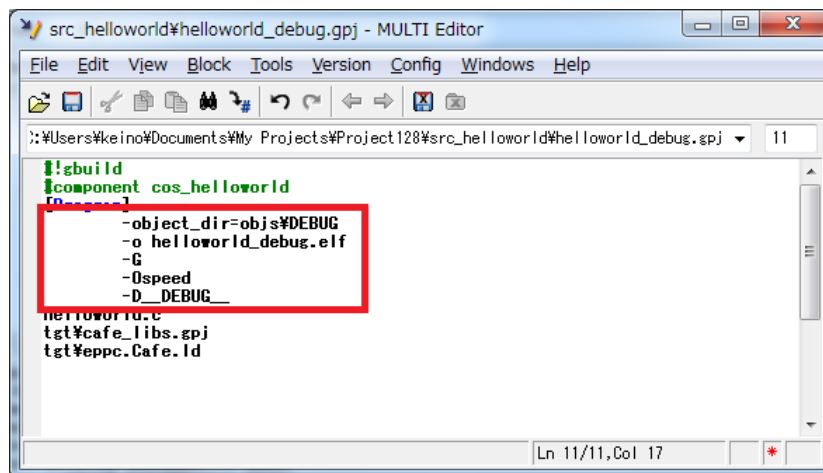
11. In the **Debugging** category, set the **Debugging Level** option to **MULTI**. This will generate source-level debugging information in GHS format. **Debugging Level** is already set to **MULTI** in the top project file (default.gpj in this example), and since the Debug project inherits from this file, you do not need to set it here. However, we recommend specifying this option explicitly in order to clarify that debugging information is being generated for the Debug project.



12. In the **Preprocessor** category, add "`__DEBUG__`" to the **Define Preprocessor Symbol** list. This defines "`__DEBUG__`" as a preprocessor symbol.



13. Editing the `helloworld_debug.gpj` file after these changes have been made shows that the following command-line options have been set.



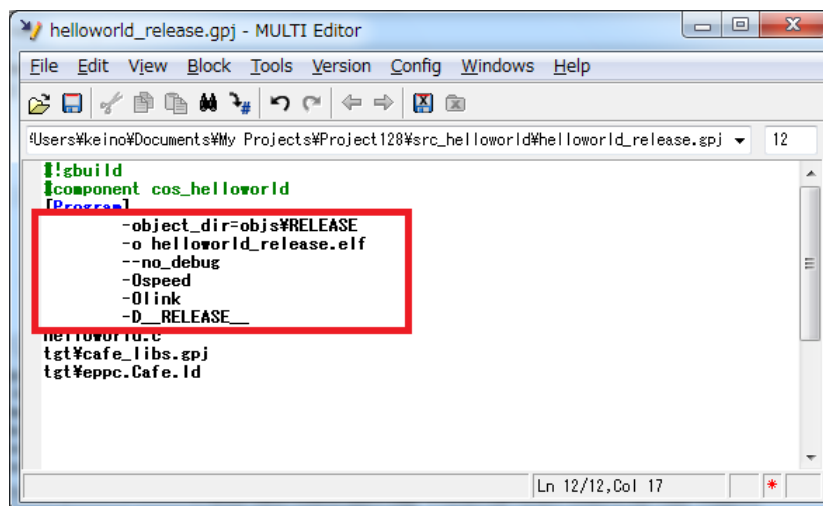
14. Configure the build options for the release project (`helloworld_release.gpj` in this example) in the same way.

In this example, we will add the following options.

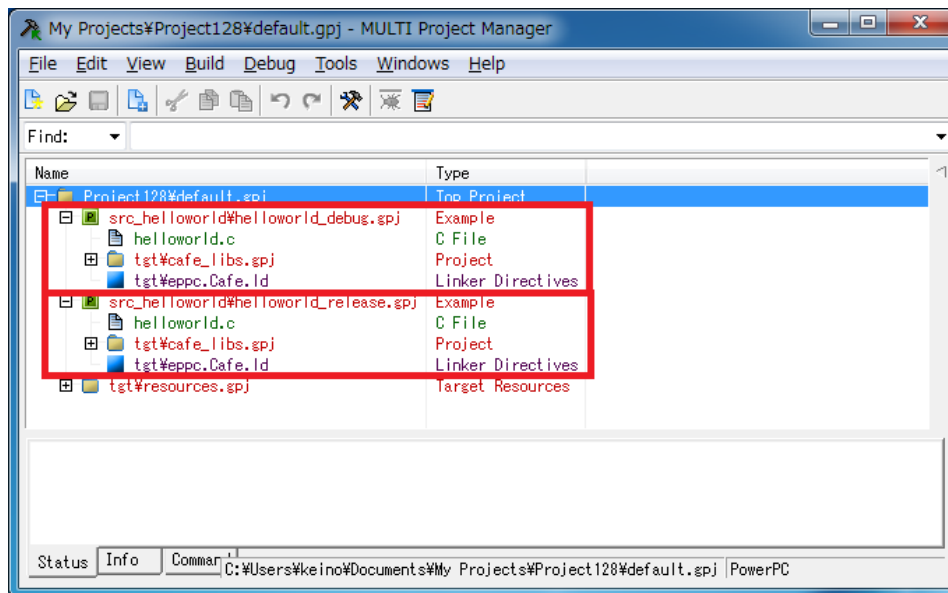
<code>-object_dir=objs\RELEASE</code>	Directory where the RELEASE objects should be created
<code>-o helloworld_release.elf</code>	Change the name of the ELF file that is generated
<code>--no_debug</code>	Don't generate debugging information
<code>-Olink</code>	Enable linker optimizations
<code>-D__RELEASE__</code>	Define the <code>__RELEASE__</code> preprocessor symbol

- To set the `--no_debug` option, open the **Build Options** dialog box, then in the **Debugging** category, set the **Debugging Level** option to **None**.
- To set the `-Olink` option, open the **Build Options** dialog box, then in the **Optimization** category, set the **Linker Optimizations** option to **On**.

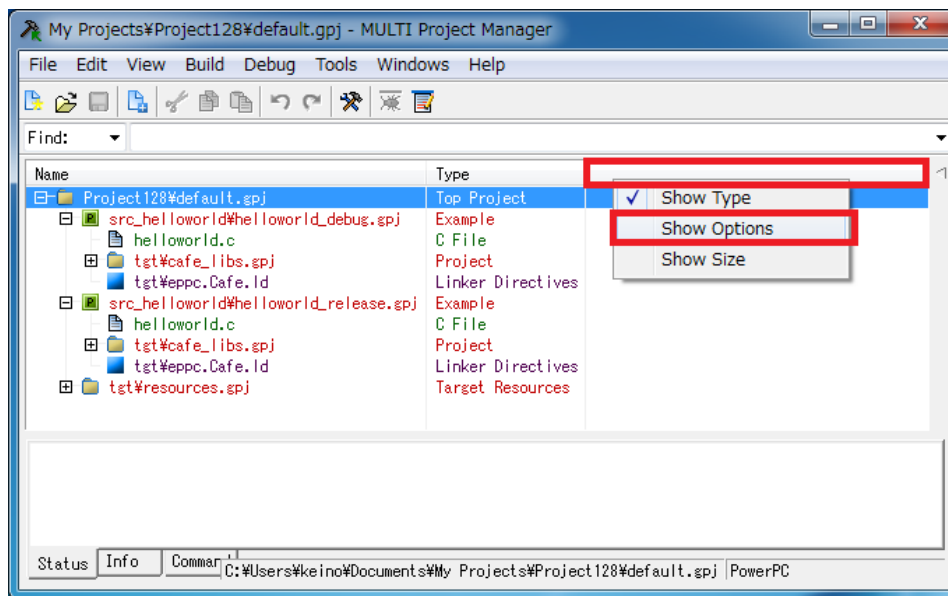
Set the other options in the same way that you did for the DEBUG project.



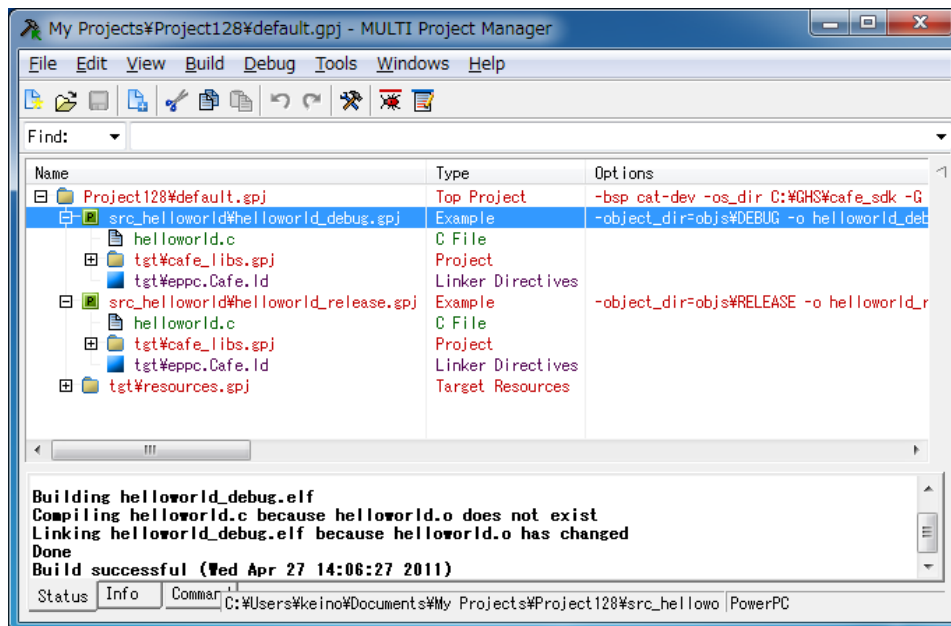
15. In the **Project Manager** window, select **File > Reload Saved Project** to apply the results of the edits you made. This allows you to manage each build as a separate program project.



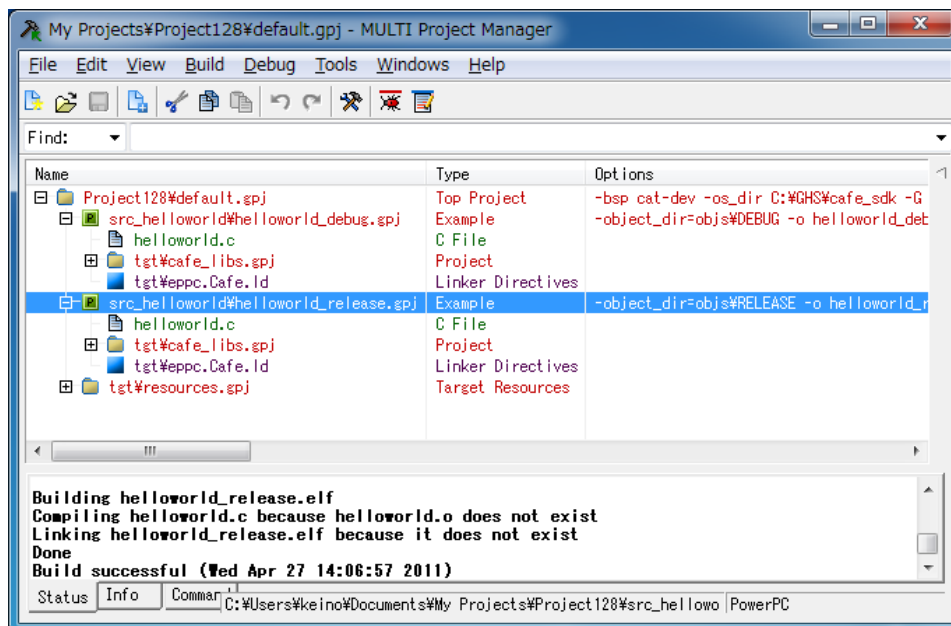
16. The **Project Manager** window doesn't show the command-line options by default. To display the options, right-click the area to the right of the **Name** and **Type** column headings, and select **Show Options**.



17. To build the *DEBUG* project, select the project for the debug program and either click the **Build** button or press F7.

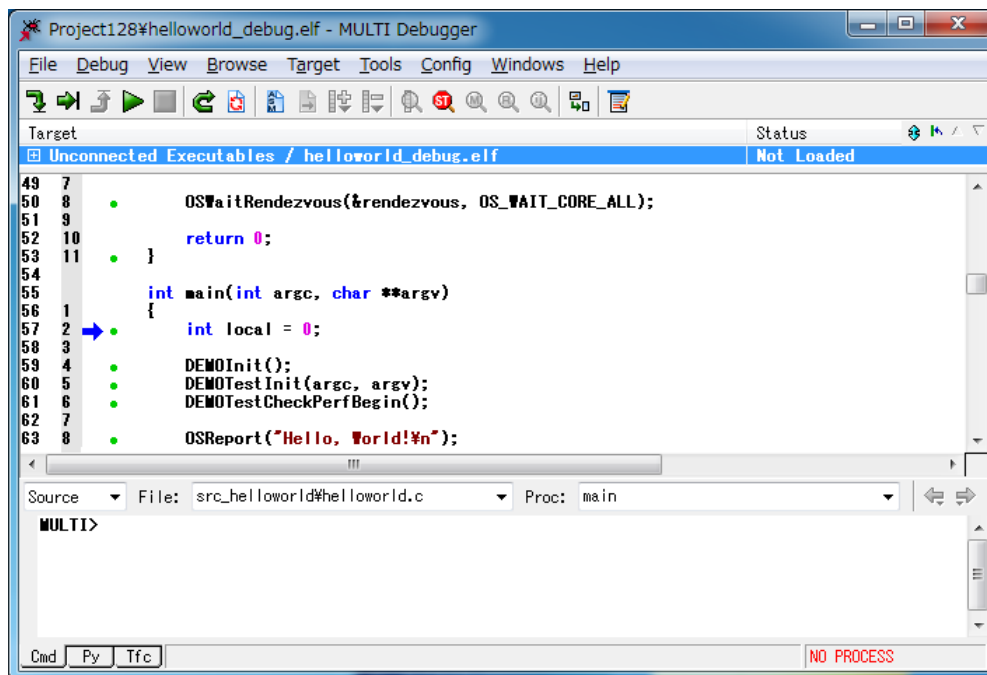


18. To build the *RELEASE* project, select the project for the release program and either click the **Build** button or press F7.

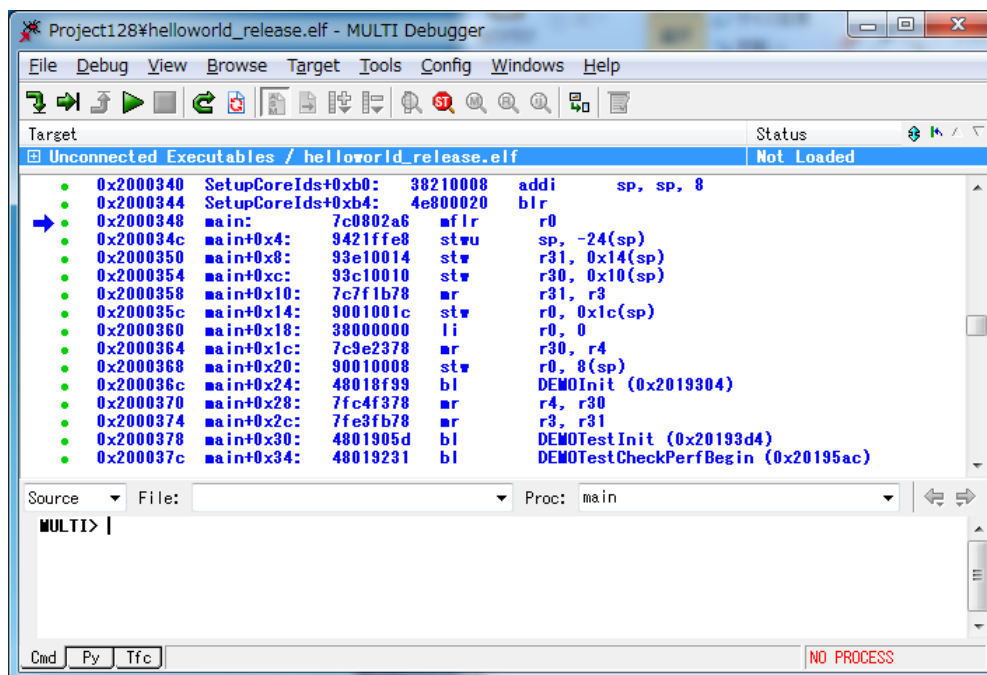


19. You can build both projects simultaneously by selecting the top project (*default.gpj* in this example) and performing a build.

20. To debug the *DEBUG* project, select the project for the debug program and either click the **Debug** button or press F5. This will launch the MULTI Debugger; debugging instructions are provided in the next chapter.



21. To debug the *RELEASE* project, select the project for the release program and either click the **Debug** button or press F5.



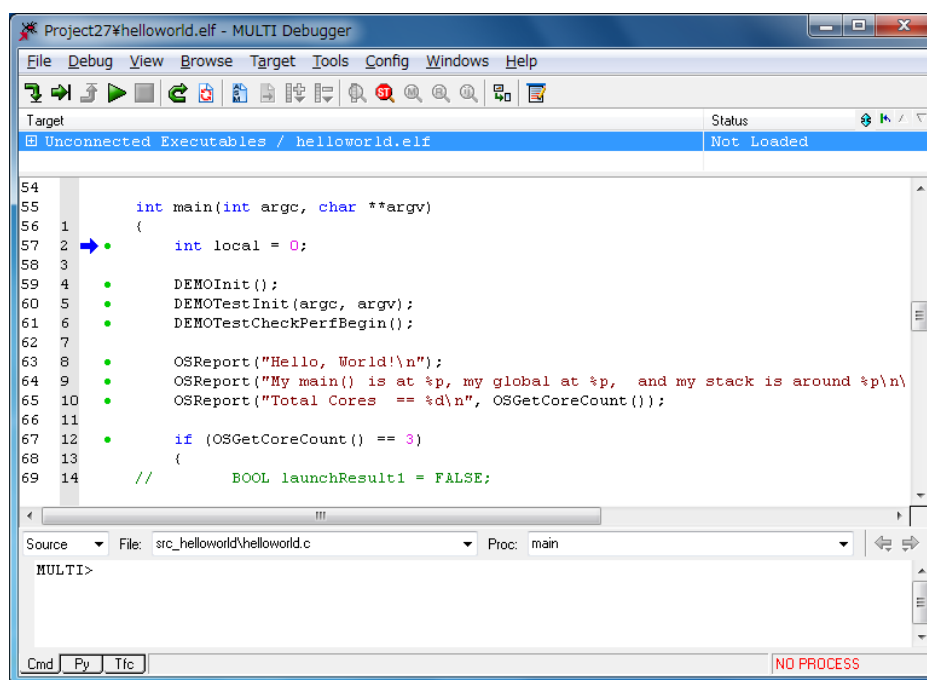
Because this project has been configured not to generate debugging information (through the `--no_debug` option), the source code is not displayed.

3 Debugging Programs

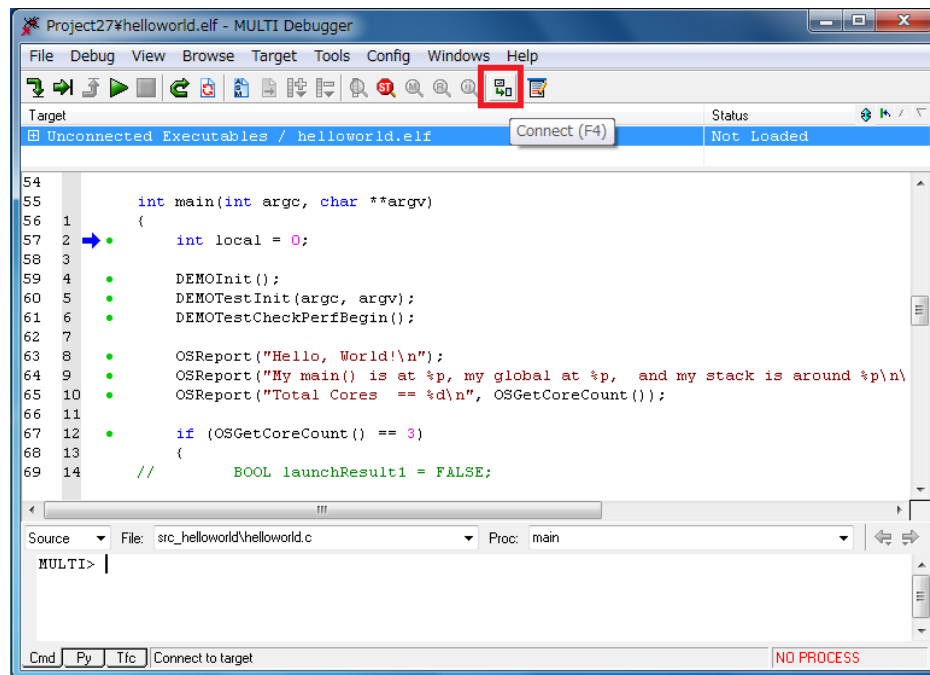
This chapter explains how to debug programs.

3.1 Launching the MULTI Debugger

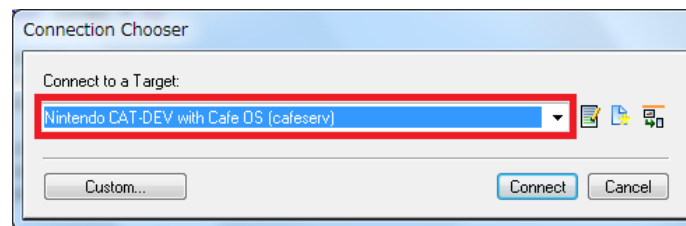
1. To launch MULTI Debugger, in the **Project Manager** window, click the **Debug** button or press F5.



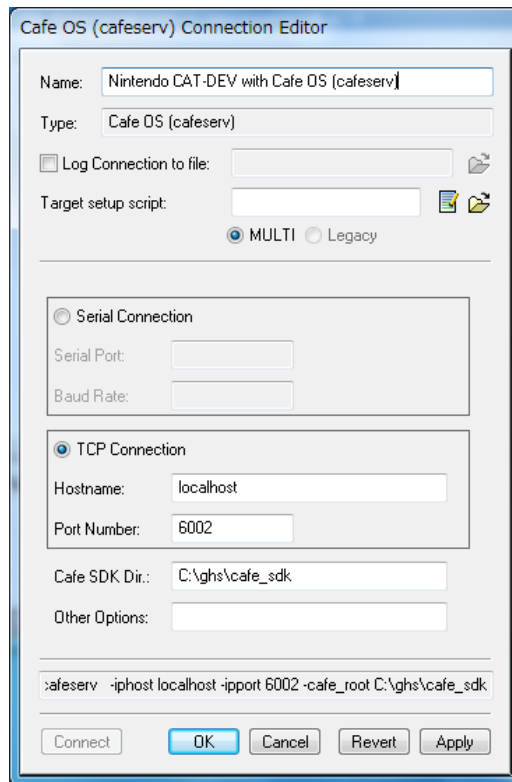
2. Click the **Connect** button or press F4 to open the **Connection Chooser**.



3. Select **Nintendo CAT-DEV with Cafe OS (cafeserv)**.

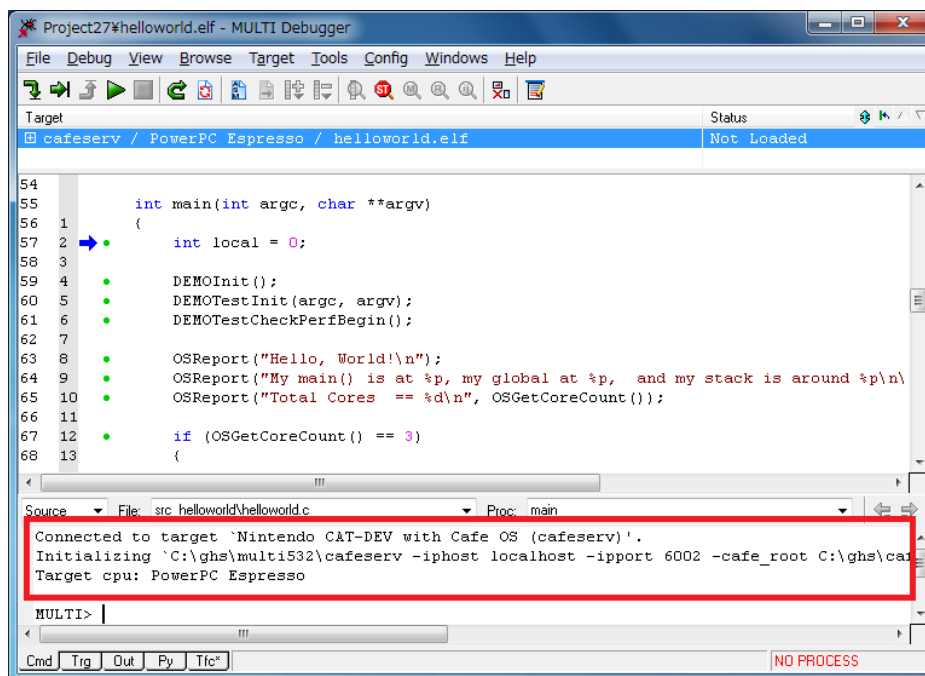


4. To change the connection settings, click the **Edit the selected Connection Method** button in the **Connection Chooser** to open the **Connection Editor** window, which allows you to make changes.



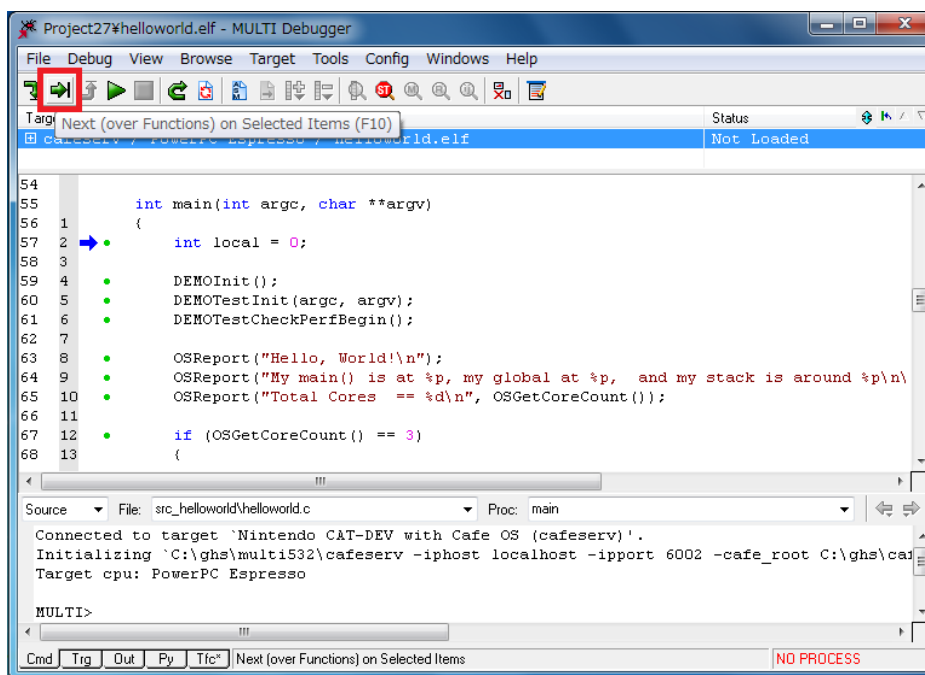
5. From the **Connection Chooser**, click the **Connect** button to connect to the target. Once the connection is established, the message shown below is displayed, and the **Connect** button will change to a **Disconnect** button.

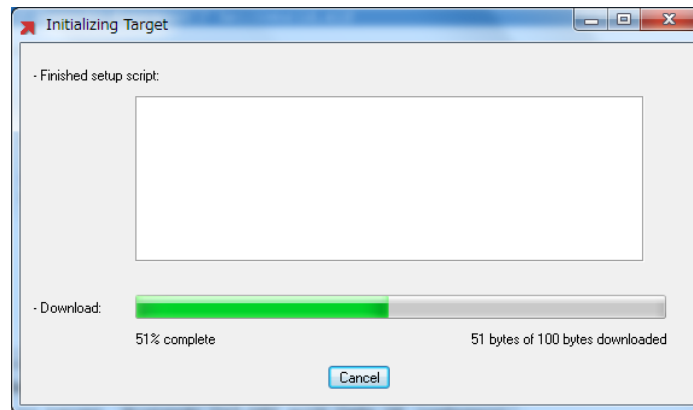
```
Connected to target 'Nintendo CAT-DEV with Cafe OS (cafeserv)'  
Initializing 'C:\GHS\multi532\cafeserv -iphost localhost -ipport 6002 -cafe_root  
C:\GHS\cafe_sdk  
Target cpu: PowerPC Espresso
```

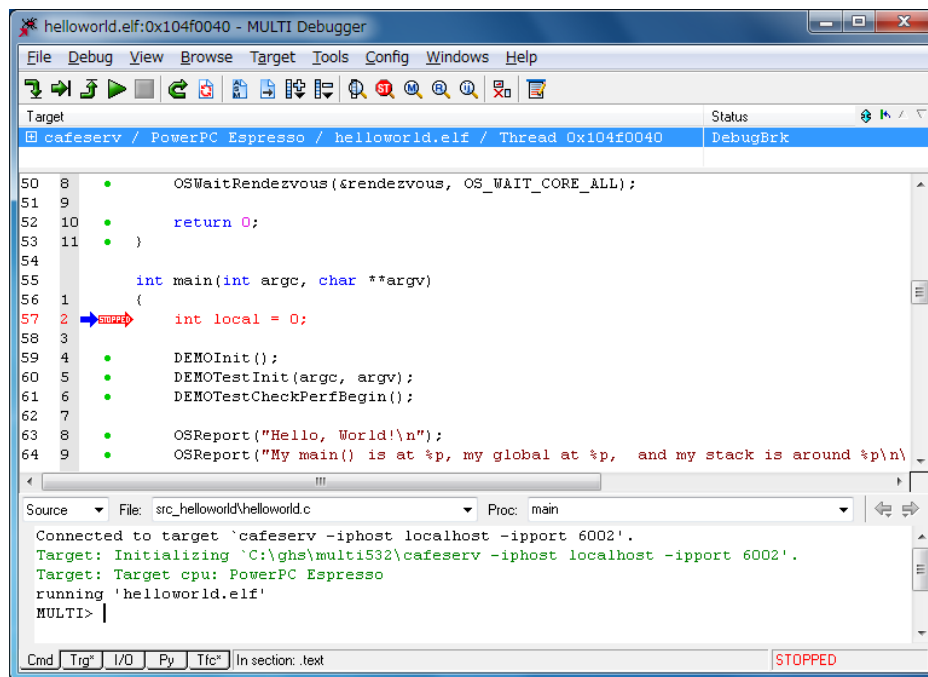
6. Perform a step operation by clicking either the **Next (over Functions) on Selected Items** button (F10) or the **Go on Selected Items** button (F5).

Note: If CAFE_ROOT is specified in the environment variable, the program will fail to load (and the 127 error is displayed.) To avoid this problem, delete CAFE_ROOT from the environment variable.





7. After stepping, the program will run up to the first line of `main()` and stop.



8. To stop debugging, simply close the debugger window. A message box will allow you to choose whether to “Quit and kill process,” “Detach from process,” or “Cancel.”

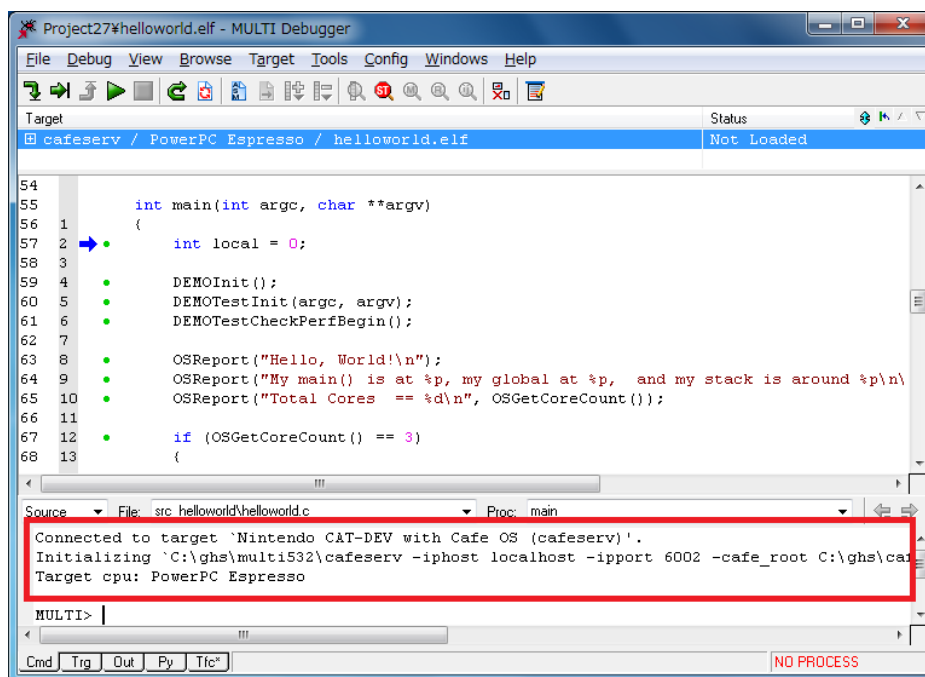
Note: As of MULTI v5.3.2 patch 1, you need to execute “cafestop” in a cygwin shell before starting up another debugging session.

3.2 Launching the MULTI Debugger from a Cygwin Shell

1. Run `cafe.bat` (located at the root directory of the Cafe SDK) to launch a Cygwin shell.
2. Change directories to the folder where the file to debug (`helloworld.elf` in this example) is located.
3. Run the following command.

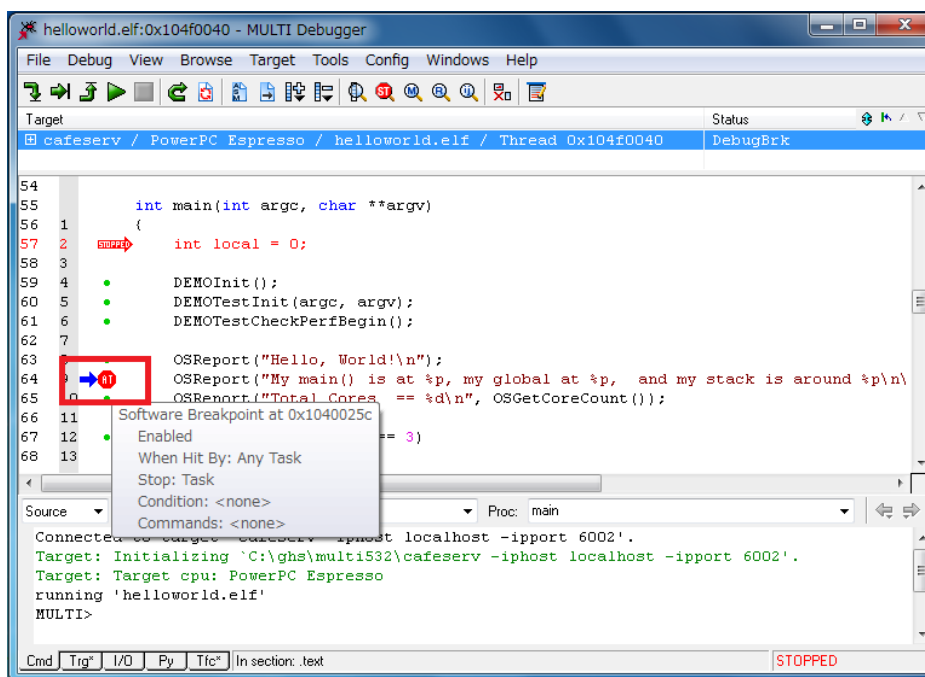
```
caferun -g 1 -d multi <program_name.elf>
```

4. The caferun command initializes the target board, then automatically launches the MULTI Debugger.

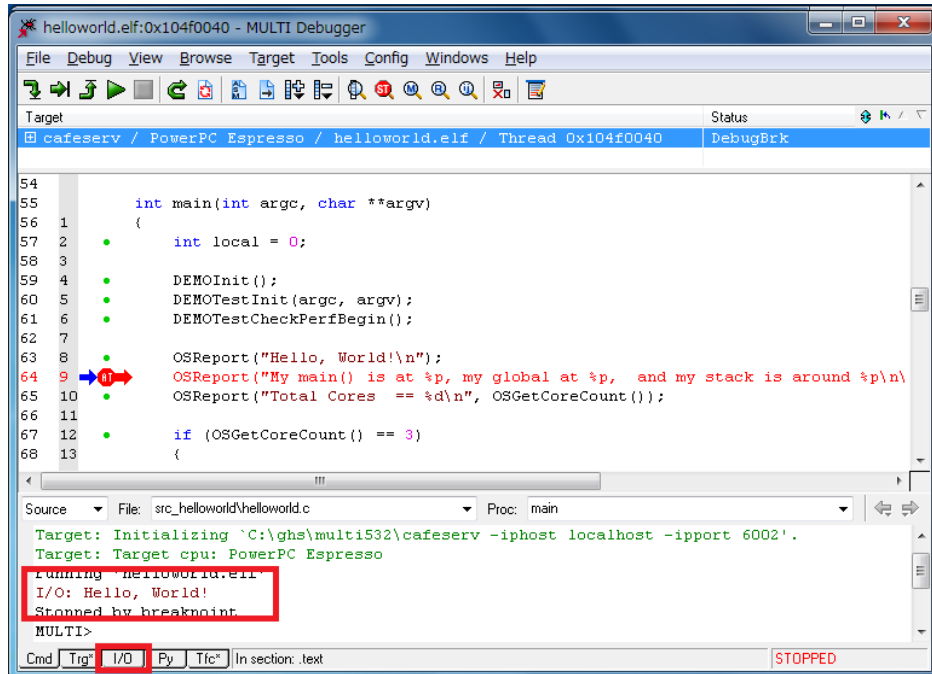


3.3 Breakpoints

1. You can set a software breakpoint by clicking on a green "breakdot."

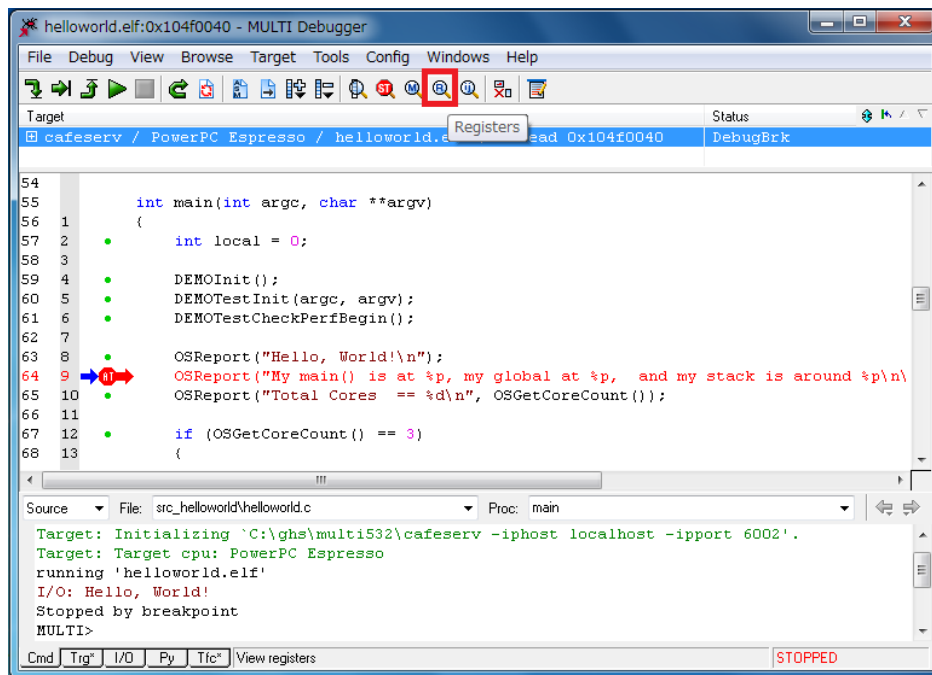


2. Clicking the **Debug** button (F5) thereafter will run the program and stop execution at the breakpoint you have set. Standard output from commands like `printf()` are displayed in the **I/O** pane.

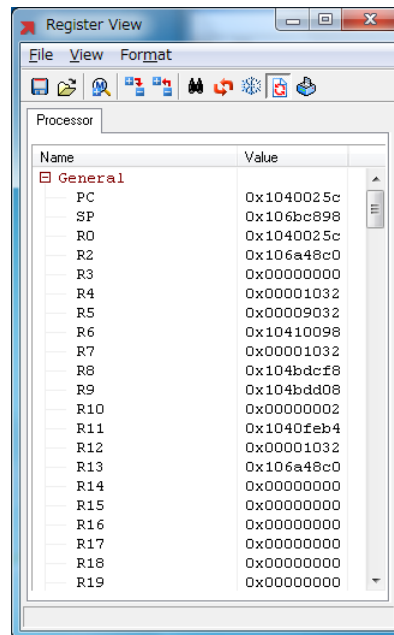


3.4 Displaying Registers

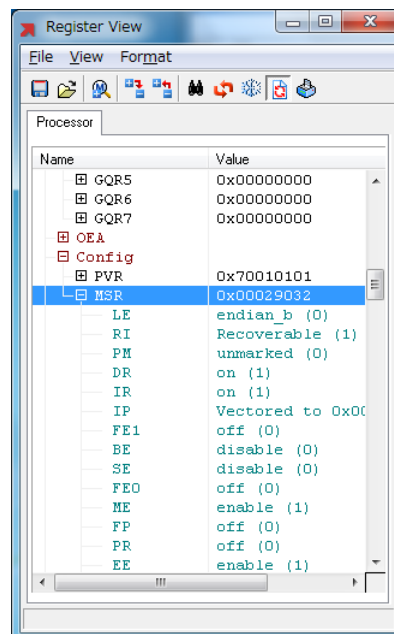
1. To check the contents of the registers, click the **Registers** button (the icon looks like the letter "R" inside a magnifying glass).

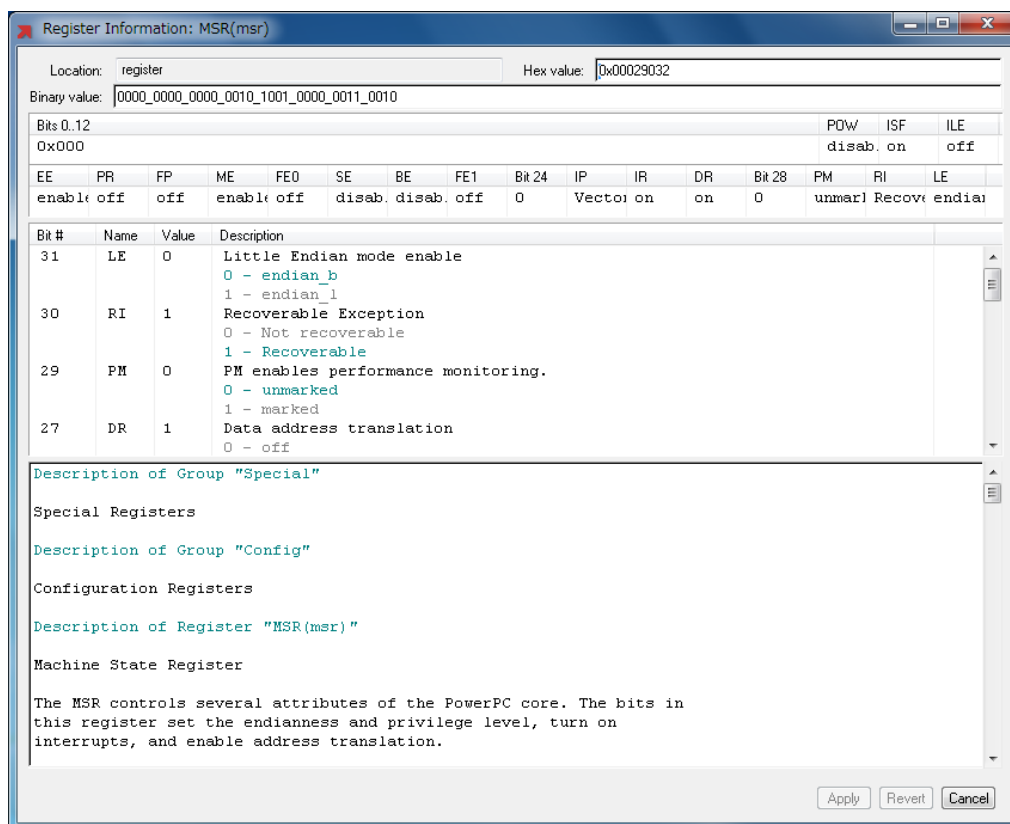


2. This opens the **Register View** window, which displays the contents of each register and allows you to change their values.



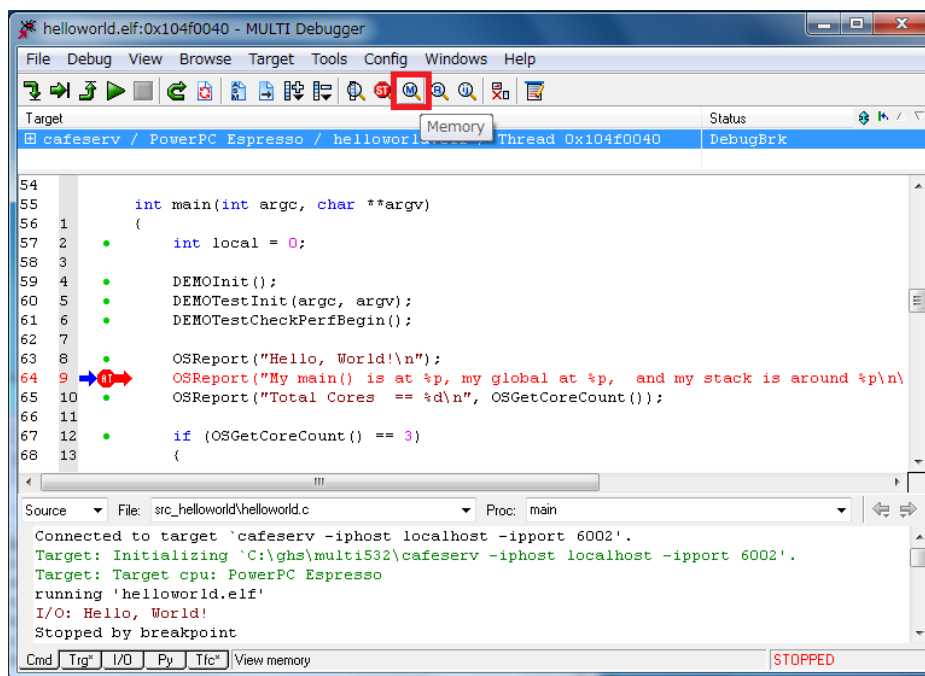
3. Double-clicking a control register (*MSR* in this example) opens the **Register Information** window, which allows you to confirm and change the flag values at the bit level.



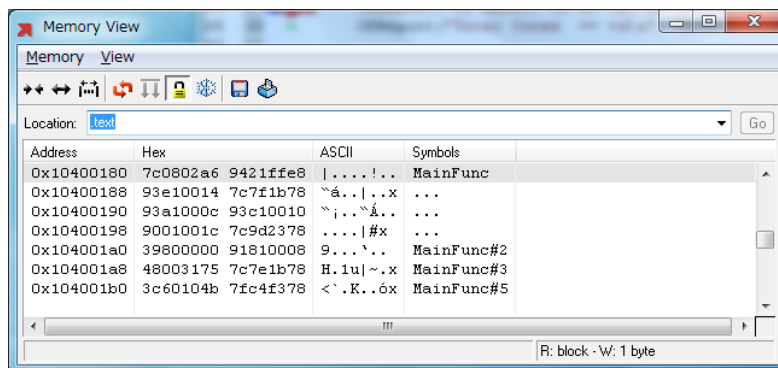


3.5 Displaying Memory

1. To check the contents in memory, click the **Memory** button (the icon looks like the letter "M" inside a magnifying glass).

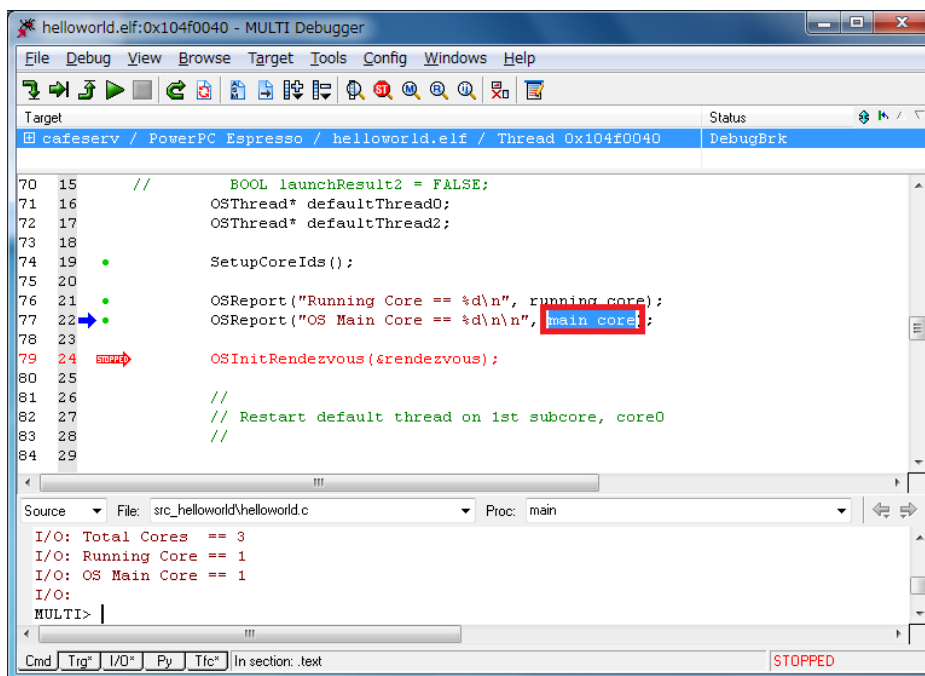


2. This opens the **Memory View** window, which allows you to confirm and change the content stored in memory, symbol names, and their values.

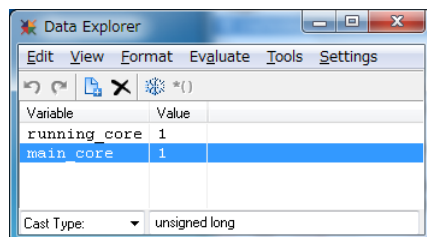


3.6 Displaying Data

1. To check the contents of variables and other such data, double-click the variable you want to display.

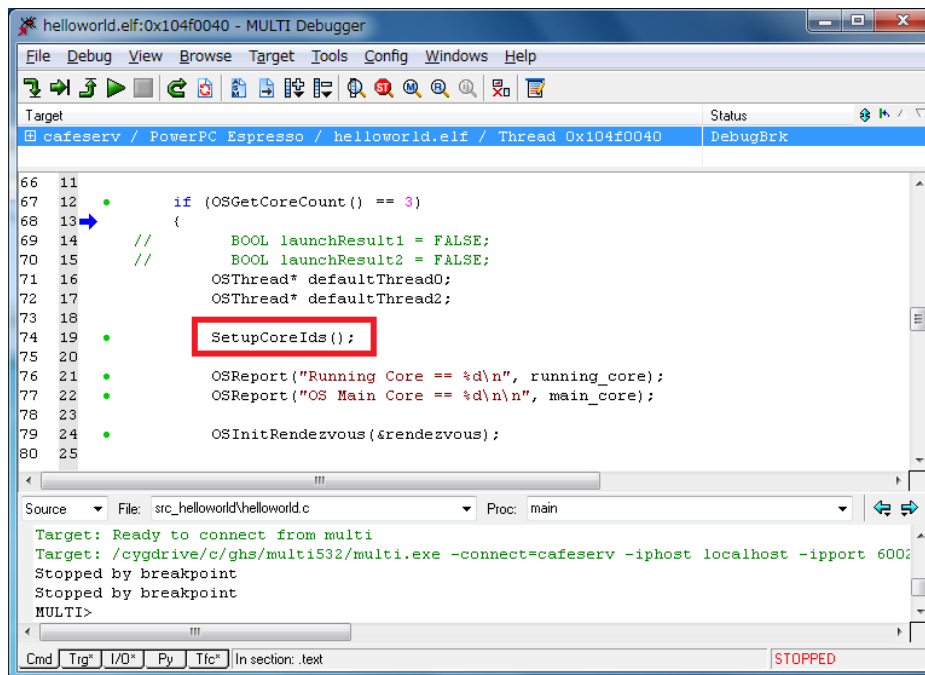


2. This opens the **Data Explorer** window, which allows you to confirm and change the types, addresses, and values of variables.

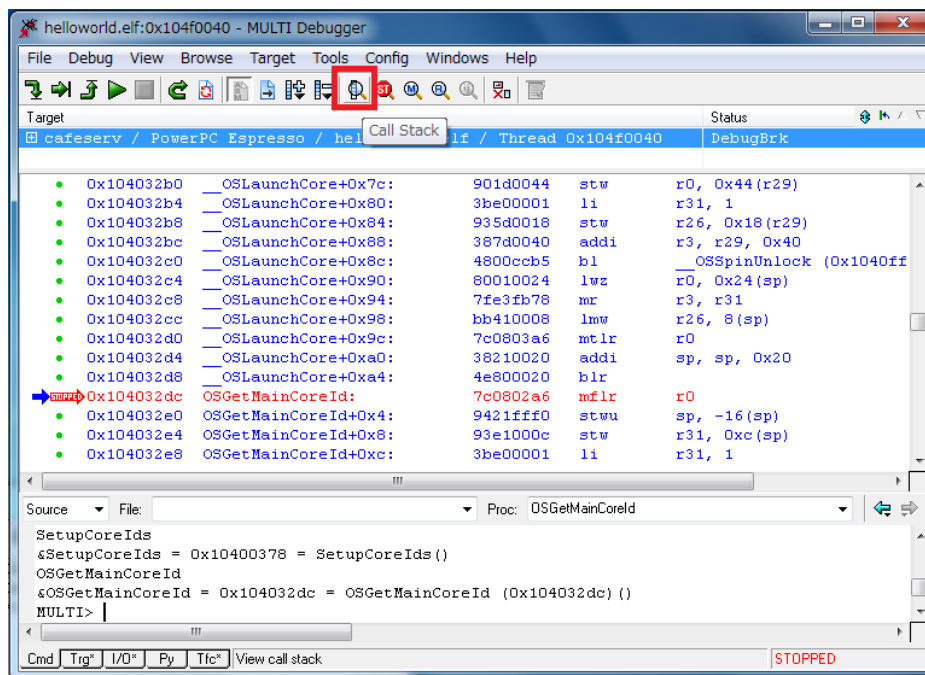


3.7 Displaying the Call Stack

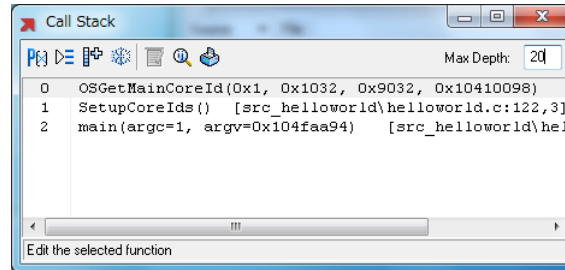
1. By single-clicking a function in the MULTI Debugger, you can trace through that function's call history in the same way as you would navigate a web browser using hyperlinks. Click the blue left-facing arrow near the lower-right corner of the window to display the current function's caller.



2. Break execution within a callee, then click the **Call Stack** button.

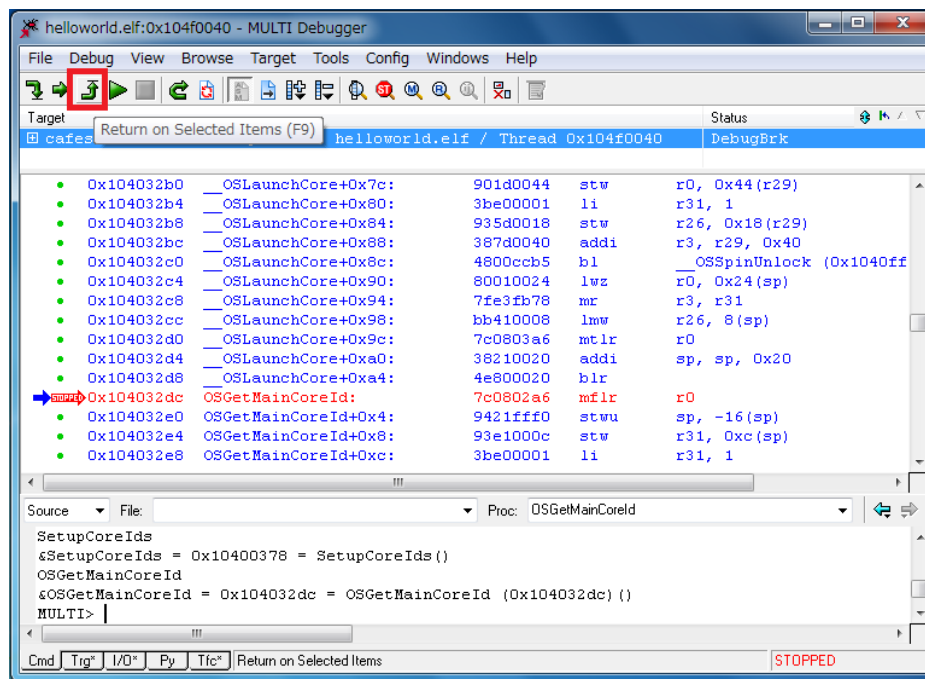


3. This will display the stack frame (function call history).



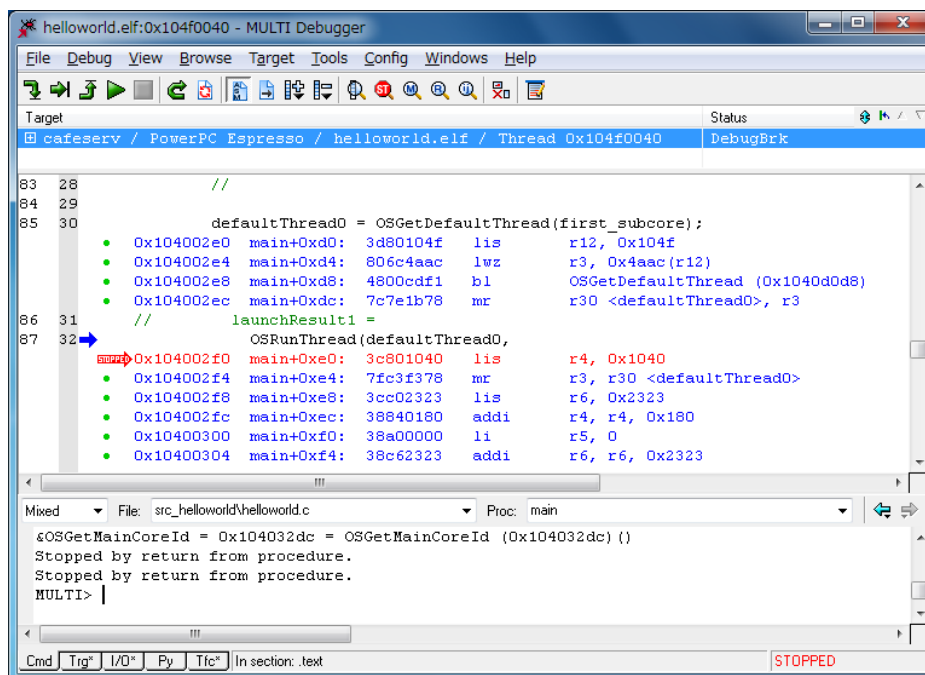
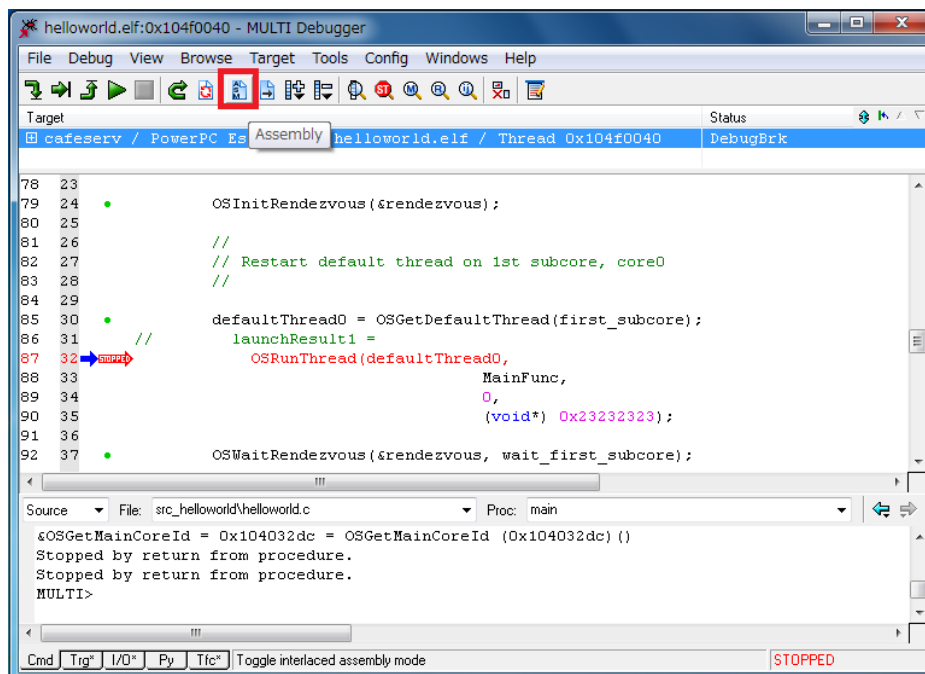
3.8 Returning from Callees

1. To return control from a callee to the caller, click the **Return on Selected Items** button (F9).



3.9 Switching Between Source and Assembly Views

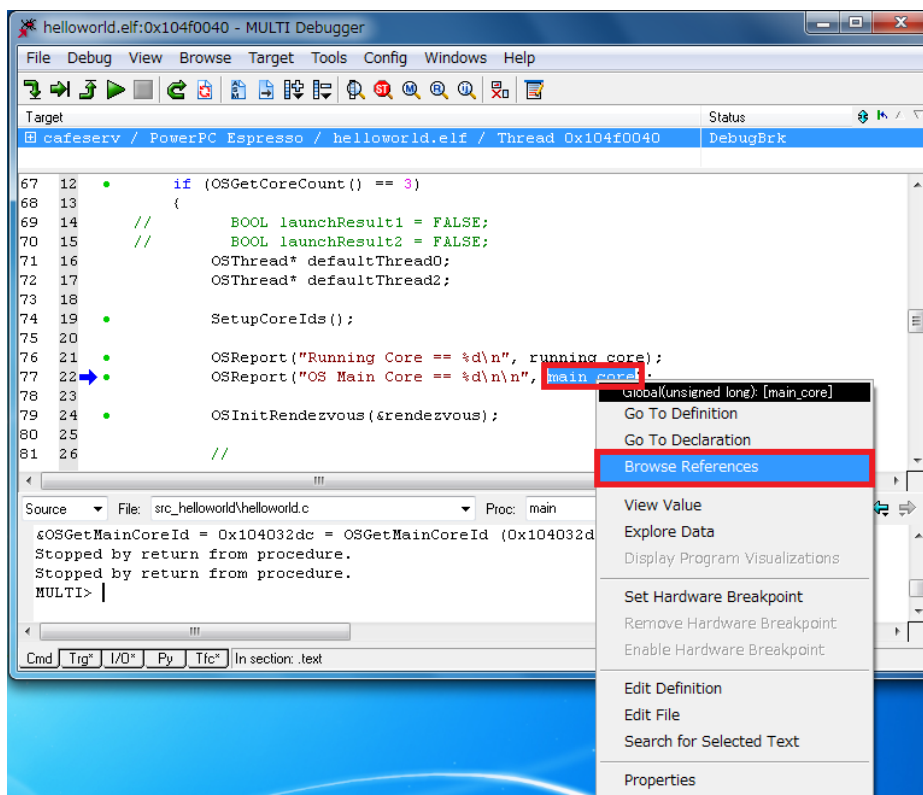
1. Click the **Assembly** button to toggle between the source only and mixed (interlaced) assembly views.



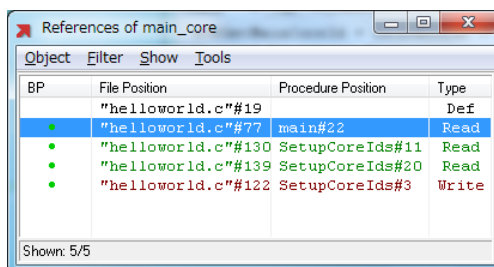
To display only assembly code, use the drop-down menu near the lower-left corner of the window to change **Mixed** to **Assembly**.

3.10 Browsing Cross References

1. To browse a symbol's cross references, right-click the symbol in the debugger's source pane, and then select **Browse References** from the shortcut menu that appears.

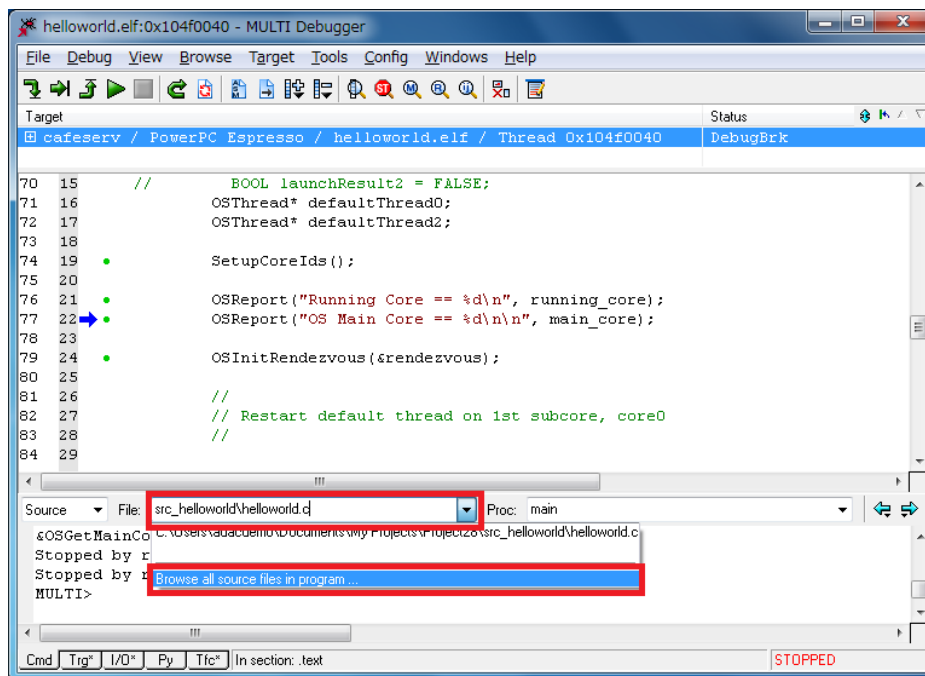


2. The **References** pane lets you see where the selected symbol is defined, shows the type of reference (for example, *Def* for definitions), and allows you to set a breakpoint at the selected position.

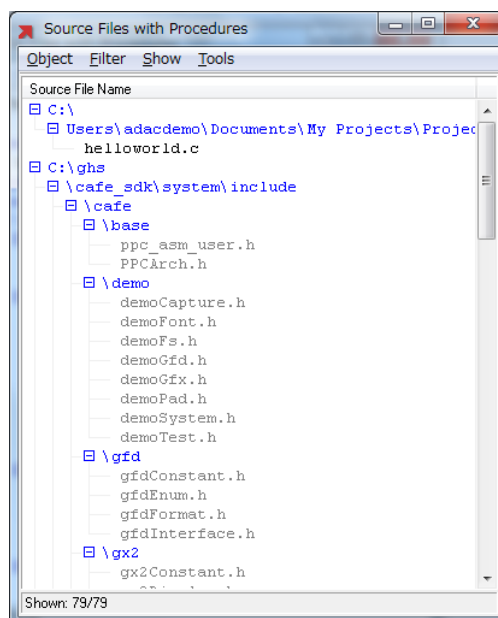


3.11 Browsing Component Source Files and Functions

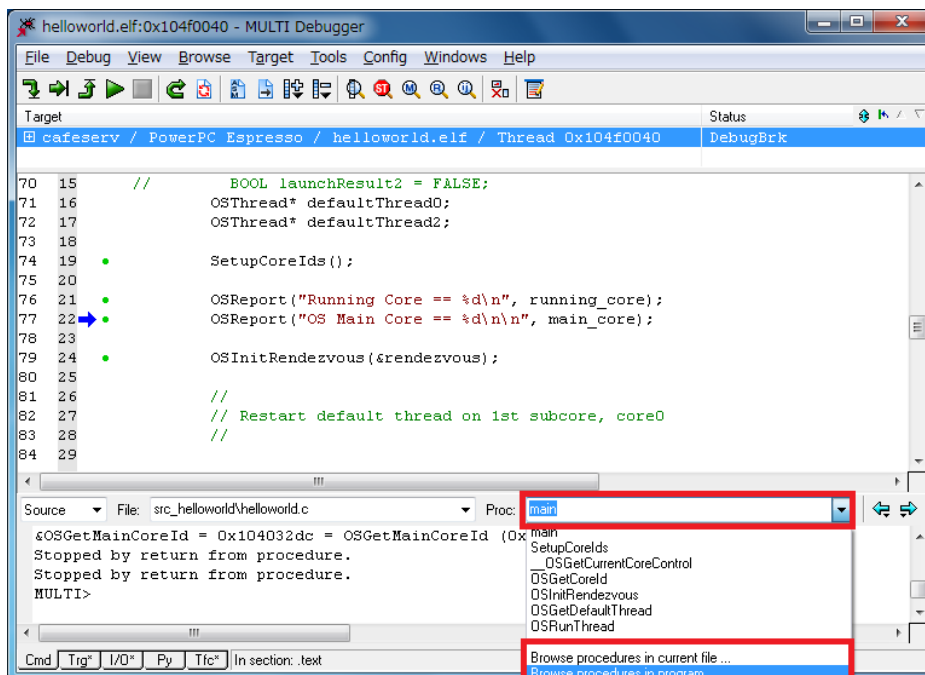
1. Select **Browse all source files in program** from the **File** drop-down menu to display a list of the source files that make up the program being debugged.



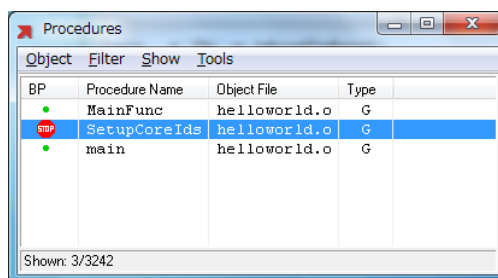
2. Click a filename within the **Source Files with Procedures** window to display the corresponding source file in the source pane.



3. Similarly, select **Browse procedures in current file** or **Browse procedures in program** from the **Proc** drop-down menu to display a list of the functions that make up the active object or program.

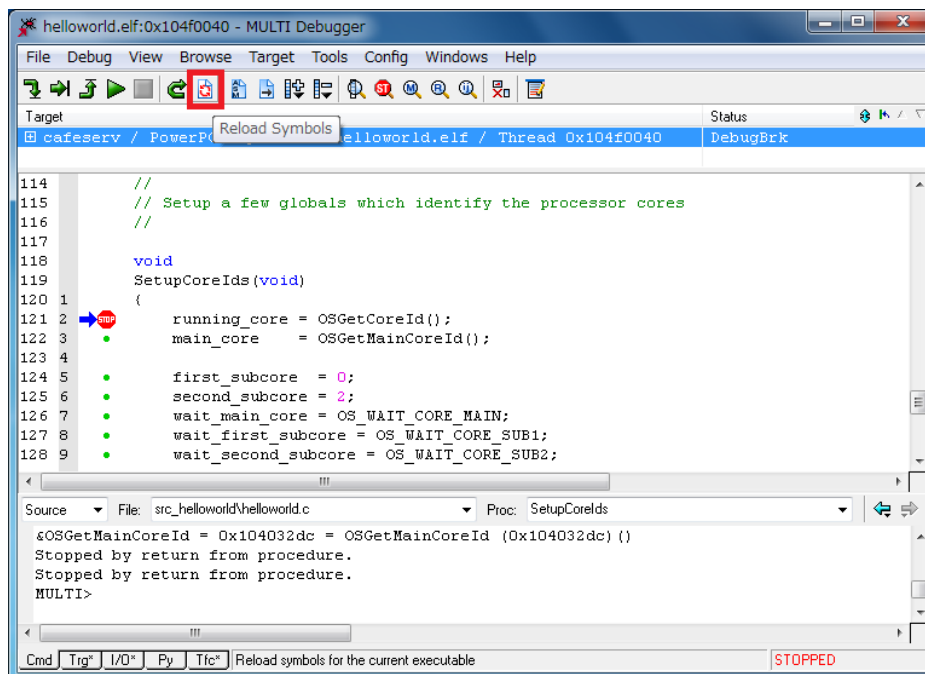


4. Select (click) the name of a function to display its source in the source pane.
Click a green breakdot within the **Procedures** window to set a breakpoint at the start of the corresponding function.

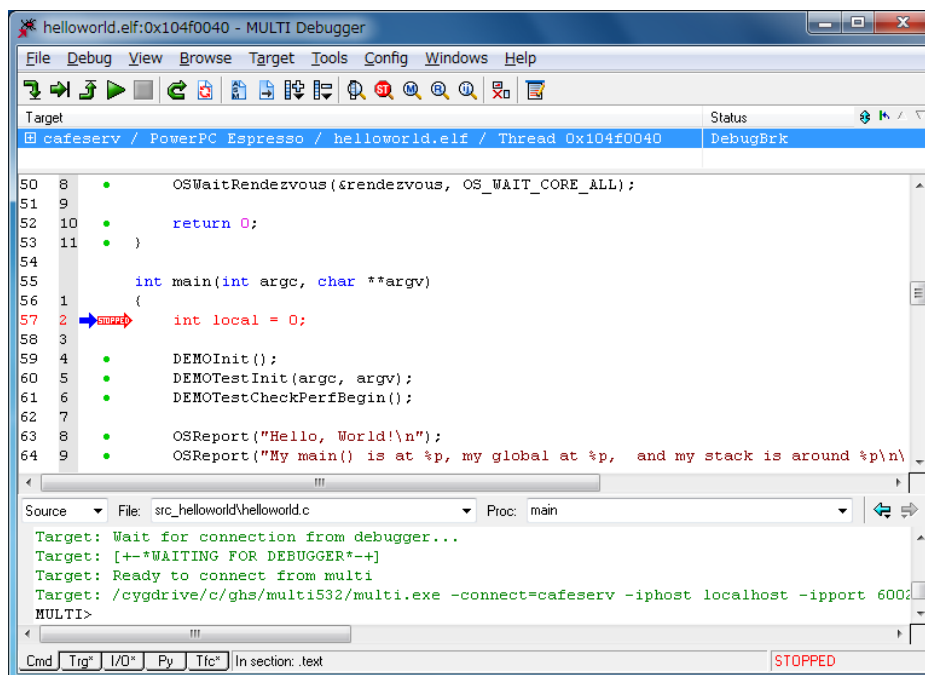


3.12 Reloading Programs

1. Click the **Reload Symbols** button to reload the current program into memory, which restores the program to the state immediately after loading.

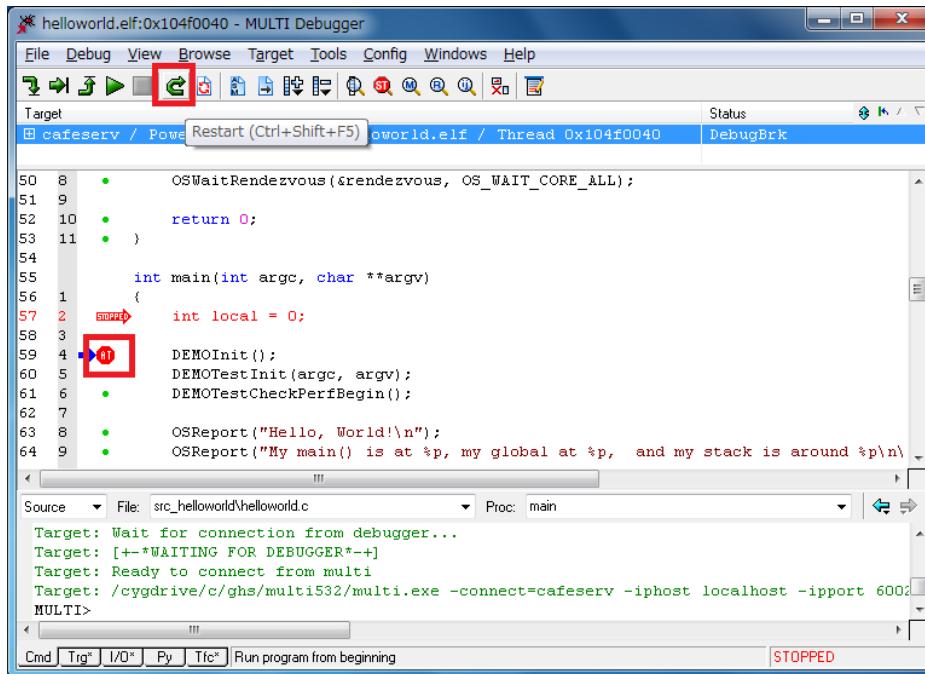


2. Step execution (press F11) to run the program up to the first line of `main()` and stop.

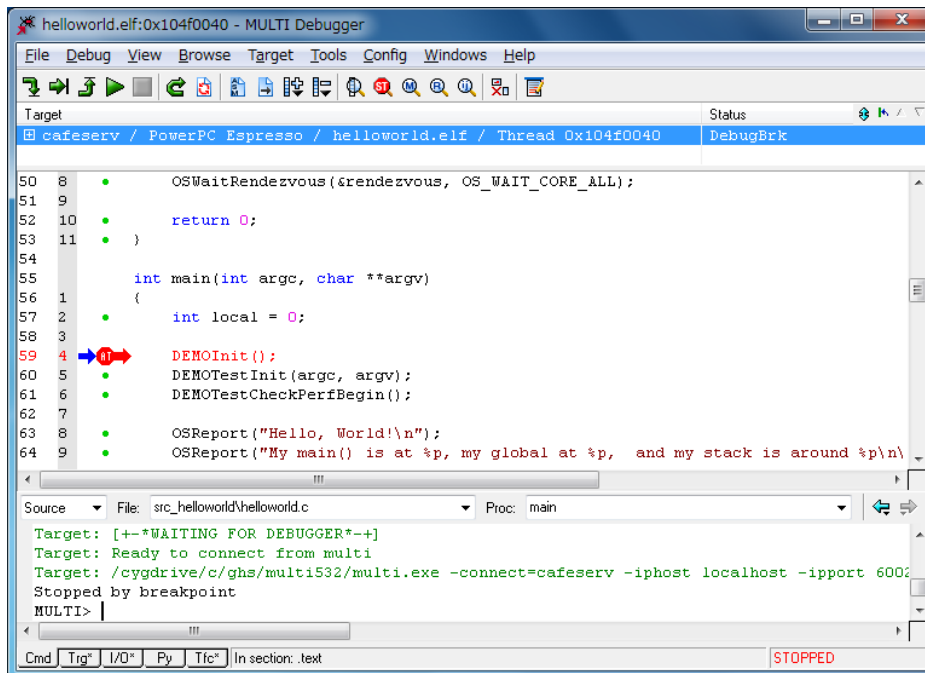


3.13 Restarting Programs

1. To restart a program, set a breakpoint where you want execution to stop, then click the **Restart** button (or press CTRL+SHIFT+F5).



2. After reloading, the program will hit the breakpoint you have set and halt execution.



Revision History

Version	Revision Date	Description
1.06	—	Added note about CAFE_ROOT in section 3.1.
1.05	—	Initial version.

All company and product names in this document are the trademarks or registered trademarks of their respective companies.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Green Hills Software, the Green Hills logo, and MULTI are trademarks of Green Hills Software, Inc.

All names of other companies and the names of their products appearing within this document are trademarks or registered trademarks of their respective owners.

The content of this document may change in the future.

The content appearing in this document is intended to provide operational examples and applications of the software. Consequently, customers using this information should be aware that they are responsible for the results. The authors of this document are not responsible for any loss or damages to the customer or any third parties that result from the use of this information.

© 2011 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed, or loaned in whole or in part without the prior approval of Nintendo.